

# Optimisation:

De l'estimation paramétrique à l'apprentissage,  
une ballade entre théorie et pratique

S. Delprat

Chapitre 7 – Apprentissage profond  
(deep learning)

## Que souhaite t on faire?

- Effectuer des régressions et/ou classification sur des ensembles données de très grande taille

typiquement : une image  $128 \times 128$  px = 16 384 données (par image)

- Traiter de très grandes quantités de données,  
typiquement 10 000 à 200 000 images voire plus pour des applications grande échèle

- Pour des réseaux de grande taille

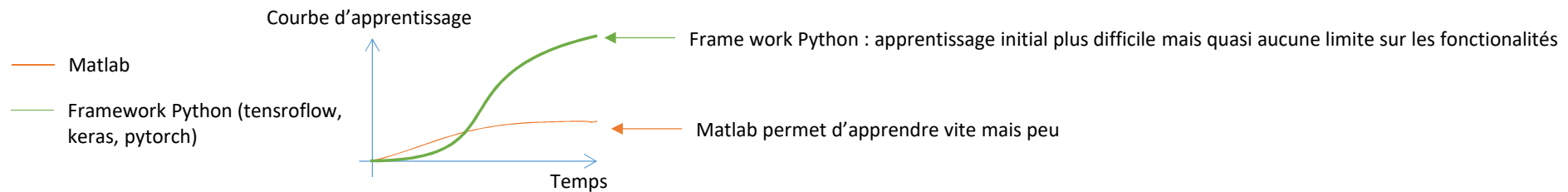
typiquement 23 millions de paramètres pour un réseau ResNet-50

# Quel framework ?

## Matlab



Uniquement pour découvrir les concepts du deep learning. Peu efficace, utilisation très marginale. Intérêt : framework simple (voire simpliste). Inconvénient: framework fermé, en général 12 mois de retard sur l'état de l'art minimum. Beaucoup de limitations.



Crée par Google, supporte C++, Python et R.

**PYTORCH** Créée par Facebook. Torch : C/C++ PyTorch: Python

# Sommaire

- Quelques rappels basiques sur le traitement d'image
  - L'algorithme du gradient stochastique
  - Préparation des données d'entrées
    - Pourquoi des reseaux à convolution ?
    - Les reseaux à convolution
      - Construction et entraînement d'un reseau avec Matlab sur la base MNIST
      - Transfert learning
      - Visualisation des réseaux
        - Architecture de quelques réseaux



### Préambule: quelques rappels succincts de traitement d'image

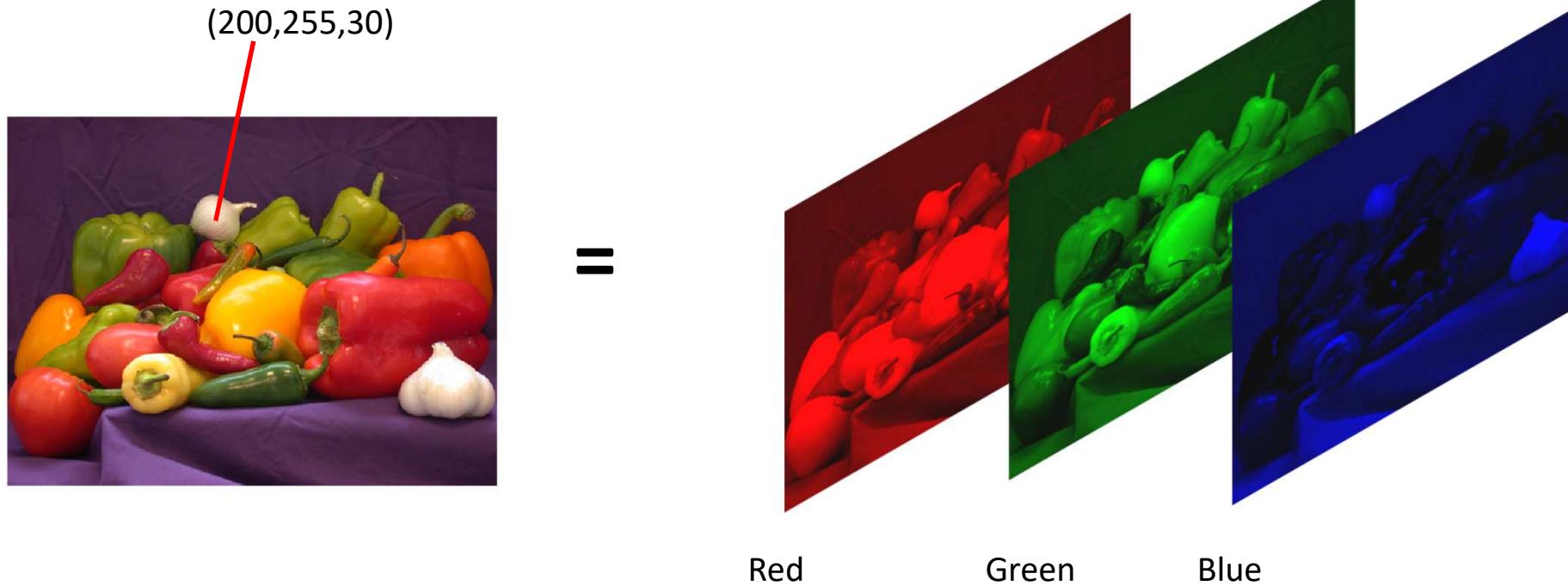


<https://deepdreamgenerator.com/>

## Deep learning – *Les images*

Une image couleur est stockée sous forme de triplets RGB

Une image couleur contient hauteur x largeur x 3 données.



# Deep learning – *Les images*

Les images sont très riches en données.

Une image couleur contient hauteur x largeur x 3 données.

Un point de l'image (pixel) est généralement codé :

- Avec 3 octets : rouge – vert – bleu (RGB) pour une image couleur
- Avec 1 octet : un niveau de gris

Une approche classique du traitement d'image consiste à utiliser des filtre pour détecter des caractéristiques et traiter l'image

L'opération de convolution consiste à déplacer le filtre sur l'image pour calculer une nouvelle image.

3	4	5	2	5	2
1	3	8	10	2	3
3	5	2	1	3	10
5	5	6	7	5	3
8	10	15	2	1	4
6	2	1	3	5	5

1	1	1
0	0	0
-1	-1	-1

$$3+4+5-3-5-2$$



2			

# Deep learning – *Les images*

Les images sont très riches en données.

Un point de l'image (pixel) est généralement codé :

- Avec 3 octets : rouge – vert – bleu (RGB) pour une image couleur
- Avec 1 octet : un niveau de gris

Une approche classique du traitement d'image consiste à utiliser des filtres pour détecter des caractéristiques et traiter l'image

L'opération de convolution consiste à déplacer le filtre sur l'image pour calculer une nouvelle image.

3	4	5	2	5	2
1	3	8	10	2	3
3	5	2	1	3	10
5	5	6	7	5	3
8	2	4	2	1	4
6	2	1	3	5	5

1	1	1
0	0	0
-1	-1	-1

$$4+5+2-5-2-1$$



2	3		



# Deep learning – *Les images*

Les images sont très riches en données.

Un point de l'image (pixel) est généralement codé :

- Avec 3 octets : rouge – vert – bleu (RGB) pour une image couleur
- Avec 1 octet : un niveau de gris

Une approche classique du traitement d'image consiste à utiliser des filtres pour détecter des caractéristiques et traiter l'image

L'opération de convolution consiste à déplacer le filtre sur l'image pour calculer une nouvelle image.

3	4	5	2	5	2
1	3	8	10	2	3
3	5	2	1	3	10
5	5	6	7	5	3
8	2	4	2	1	4
6	2	1	3	5	5

1	1	1
0	0	0
-1	-1	-1

$$5+2+5-2-1-3$$



2	3	6	

# Deep learning – *Les images*

Les images sont très riches en données.

Une image couleur contient hauteur x largeur x 3 données.

Un point de l'image (pixel) est généralement codé :

- Avec 3 octets : rouge – vert – bleu (RGB) pour une image couleur
- Avec 1 octet : un niveau de gris

Une approche classique du traitement d'image consiste à utiliser des filtres pour détecter des caractéristiques et traiter l'image

L'opération de convolution consiste à déplacer le filtre sur l'image pour calculer une nouvelle image.

3	4	5	2	5	2
1	3	8	10	2	3
3	5	2	1	3	10
5	5	6	7	5	3
8	2	4	2	1	4
6	2	1	3	5	5

1	1	1
0	0	0
-1	-1	-1

7+5+3-3-5-5



2	3	6	-5
-4	3	2	0
-4	0	-1	7
7	12	9	2

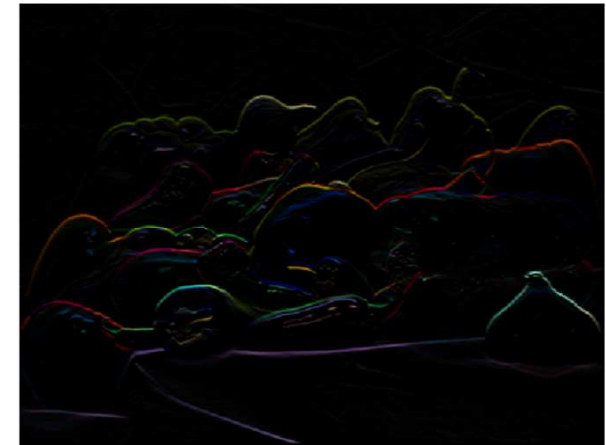
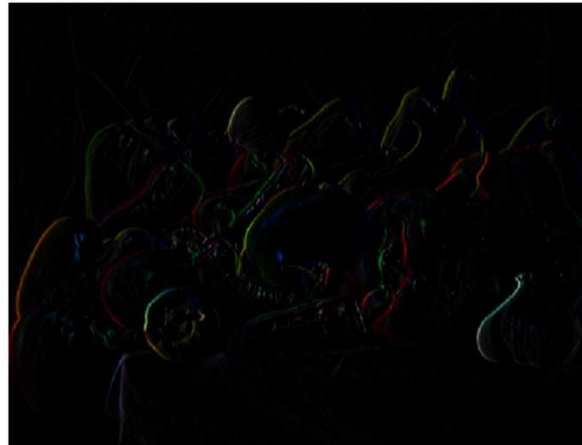
## Deep learning – *Les images*

En traitement d'image « classique », on applique le même filtre au 3 couches, ce n'est cependant pas obligatoire.

Exemple : détection de gradients horizontaux & verticaux avec un filtre de Prewitt

$$\text{Prewit} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} / 3$$

$$\text{Prewit} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} / 3$$



# Deep learning – *Les images*

```
clear all
close all;
clc

originalRGB = imread('peppers.png');
imshow(originalRGB);
c=1;
Prewit=[-1 0 1;
        -1 0 1;
        -1 0 1]/3;
filteredRGB2 = imfilter(originalRGB, Prewit);
filteredRGB3 = imfilter(originalRGB, Prewit');

figure,
subplot(1,3,1);
imshow(originalRGB)
subplot(1,3,2);
imshow(filteredRGB2)
subplot(1,3,3);
imshow(filteredRGB3)
```

fspecial('unsharp')



Original



fspecial('disk',10)



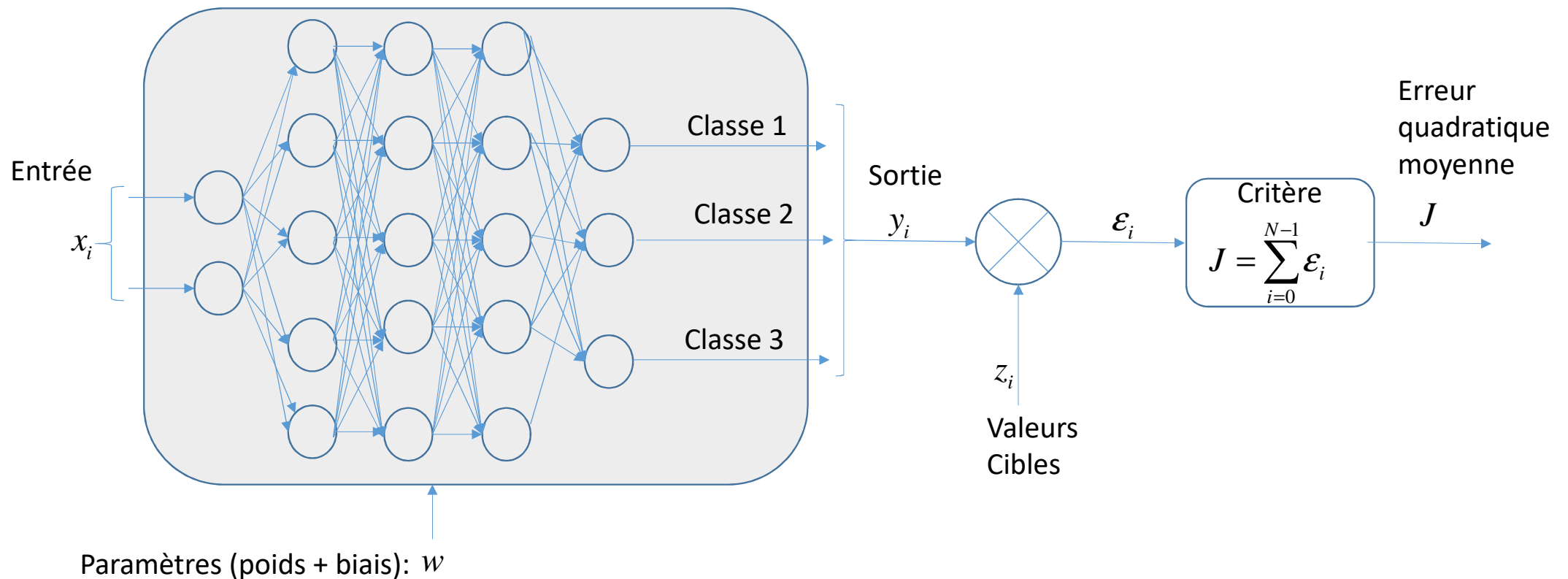
# Sommaire

- Quelques rappels basiques sur le traitement d'image
- **L'algorithme du gradient stochastique**
  - Préparation des données d'entrées
    - Pourquoi des reseaux à convolution ?
    - Les reseaux à convolution
      - Construction et entrainement d'un reseau avec Matlab sur la base MNIST
      - Transfert learning
      - Visualisation des reseaux
        - Architecture de quelques reseaux



# Deep learning – *Stochastic Gradient*

Que ce soit pour un problème de régression ou classification, l'apprentissage consiste à minimiser un critère (🇬🇧 loss):



## Deep learning – *Stochastic Gradient*

Que ce soit pour un problème de régression ou classification, l'apprentissage consiste à minimiser un critère (🇬🇧 loss):

$$\min_w J = \sum_{i=0}^{N-1} (z_i - f(w, y_i))^2$$

La minimisation du critère se fait par un algorithme de type descente de gradient, alors à chaque iteration  $j$  l'algorithme du gradient donne:

$$w_{j+1} = w_j - \alpha \nabla_w J(w_j)$$

↑  
Taux d'apprentissage (🇬🇧 Learning rate)

Le gradient se calcule à partir du critère:

$$\nabla_w J(w) = -2 \sum_{i=0}^{N-1} \underbrace{(z_i - f(w, y_i))}_{\text{Passe directe}} \underbrace{\frac{\partial f(w, y_i)}{\partial w}}_{\text{f : fonction composée par les différentes couches}}$$

Passe directe  
(🇬🇧 forward)

f : fonction composée par les différentes couches  
Calcul des dérivées par la règle de dérivation en chaîne  
(rétropropagation du gradient de l'erreur)

## Deep learning – *Stochastic Gradient*

Problème : Pour calculer le gradient exact, il faut calculer :

- L'erreur pour toutes les observations (i.e. les 200 000 images)
- La dérivée partielle par rapport au nombreux paramètres, pour chaque image!

$$\nabla_w J(w) = -2 \sum_{i=0}^{N-1} (z_i - f(w, y_i)) \frac{\partial f(w, y_i)}{\partial w}$$

En pratique, ce calcul n'est pas faisable:

- Trop long
- Probablement instable numériquement

Solution : une approche statistique

On suppose que les observations  $(x_i, y_i)$  sont issues de réalisations aléatoires qui suivent une loi  $P(x|y)$  inconnue

Si on sélectionne aléatoirement  $n \rightarrow \infty$ , observations pour calculer le gradient, alors l'algorithme du gradient permet d'adapter les paramètres dans la bonne direction

Pour  $n < N$ , on montre que l'algorithme converge (sous certaines hypothèses).



# Deep learning – *Stochastic Gradient*



Idée:

- + on découpe le jeu de données en mini-lots
- + on applique l'algorithme du gradient (avec moment) sur chaque mini-lot

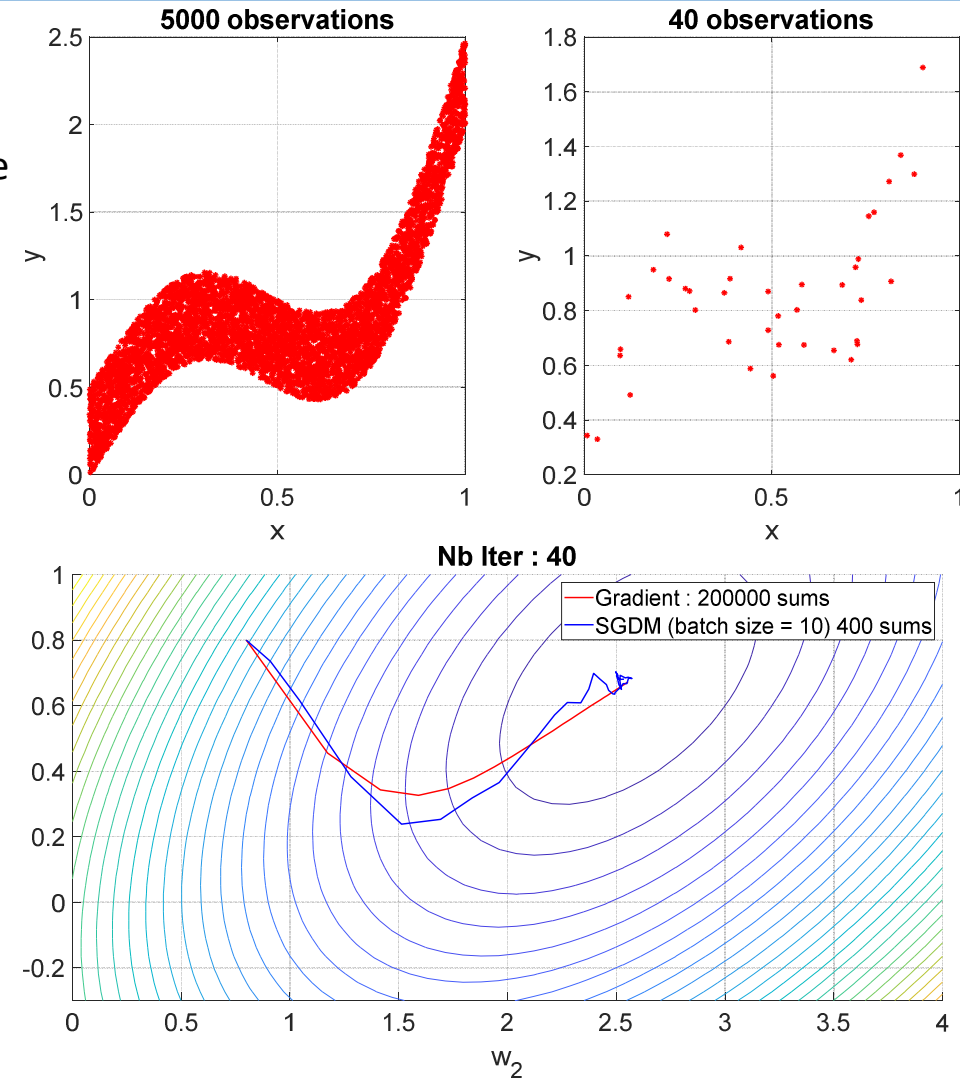
Exemple: régression  $y = w_1 \cdot x^2 + w_2 \cdot \sin(2\pi x)$

Gradient classique avec 5000 points requiert 200 000 calculs

SGDM avec mini lots de taille 10 : requiert uniquement 400 calculs

=> Au cours des itérations, des mini-lots de taille réduite apportent statistiquement la même quantité d'information mais bruitée

Une « époque » consiste à itérer l'algorithme sur tous les mini-lots qui composent l'ensemble des observations.



# Deep learning – *Stochastic Gradient*

- Méthode de rétropropagation du gradient de l'erreur « **stochastique** »  
Chaque itération n'utilise qu'**une partie** la base d'images d'entraînement



## Idée de base:

On suppose que l'ensemble des images d'entrée est issue d'une distribution

On suppose que chaque mini-lot permet d'échantillonner la distribution des données de départ

La rétro propagation est effectuée à chaque mini-lot:

- Un mini lot requiert moins de mémoire (quelques Mo vs quelques Go pour la base de départ)
- Moins de calcul par mini-lots
- Une époque correspond au traitement de tous les mini-lots composants l'ensemble des données de départ
- En général, on augmente la taille du mini-lot en fonction de la mémoire disponible sur le GPU

## Deep learning – *Stochastic Gradient*

- Modification de l'algorithme d'apprentissage

Adaptation des poids avec la méthode du gradient:

$$W_{k+1} = W_k - \alpha \cdot \nabla E(W_k)$$

→ Gradient de l'erreur par rapport aux paramètres

→ Taux d'apprentissage (  learning rate)  
=> Taille du « Pas » dans la direction du gradient

Pour limiter les oscillations des paramètres autour d'un minimum local, ils est possible de modifier l'algorithme du gradient:

$$W_{k+1} = W_k - \alpha \cdot \nabla E(W_k) - \lambda (W_k - W_{k-1})$$

→ Permet d'amortir les oscillations  
 $\lambda \in [0,1]$  par exemple 0,8

# Sommaire

- Quelques rappels basiques sur le traitement d'image
- L'algorithme du gradient stochastique
- Préparation des données d'entrées
  - Pourquoi des reseaux à convolution ?
  - Les reseaux à convolution
    - Construction et entrainement d'un reseau avec Matlab sur la base MNIST
    - Transfert learning
    - Visualisation des reseaux
      - Architecture de quelques reseaux



# Deep learning – *Normalization et/ou standardisation des données d'entrée*

Les données d'entrées peuvent être relativement quelconque. Leur échèle peuvent être très différentes.

Cela peut causer des problèmes d'instabilité numérique dans les algorithmes d'apprentissage (explosion des gradients)

Données brutes

Nombre de kilomètre parcouru  
par an : entre 0 et 20 000 km

0 20 000  
→

Nombre de passagers dans la  
voiture : entre 1 et 5

1 5  
→

Nombre de kilomètre  
parcours par semaine: entre  
0 et 1000 km

0 10 000  
→

# Deep learning – *Normalization et/ou standardisation des données d'entrée*

Les données d'entrées peuvent être relativement quelconque. Leur échèle peuvent être très différentes.

- La standardisation consiste à mettre à l'échelle les données de sorte que leur moyenne soit 0 et l'écart type 1

NB : Cela n'implique en rien que la distribution des données soit normale !!

- La normalisation consiste à mettre à l'échelle les données de sorte qu'elles soient comprise entre -1 et 1 (ou 0 et 1)

Nombre de kilomètre parcouru  
par an : entre 0 et 20 000 km



Nombre de passagers dans la  
voiture : entre 1 et 5



Nombre de kilomètre  
parcourus par semaine: entre  
0 et 1000 km



# Deep learning – *Augmentation du nombre d'image d'entrées*

Un des problèmes avec l'apprentissage des poids est que le nombre d'images classifiées disponible est limité.

En général, plus on dispose d'images classifiées et plus l'apprentissage est performant.

Une astuce qui permet d'augmenter artificiellement le nombre d'image consiste à appliquer des opérateurs sur les images d'origines qui ne modifient pas la classification:

- Symétrie
- Rotation
- Translation
- Augmentation de contraste et/ou luminosité
- Etc.

```
augmenter = imageDataAugmenter( ...  
'RandXReflection',true, ...  
'RandXScale',[1 2], ...  
'RandYReflection',true, ...  
'RandYScale',[1 2])
```

```
source = augmentedImageSource(imageSize,ImgOrigine,LabelImgOrigine,'DataAugmentation',augmenter)
```

# Sommaire

- Quelques rappels basiques sur le traitement d'image
- L'algorithme du gradient stochastique
- Préparation des données d'entrées
  - Pourquoi des reseaux à convolution ?
- Les reseaux à convolution
  - Construction et entrainement d'un reseau avec Matlab sur la base MNIST
  - Transfert learning
  - Visualisation des reseaux
    - Architecture de quelques reseaux

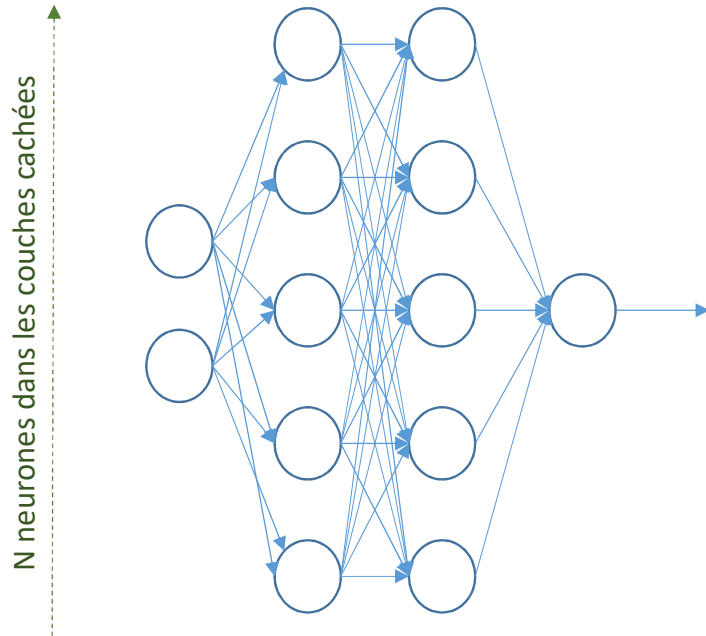




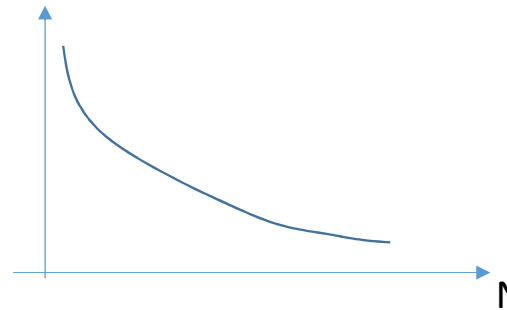
# Deep learning – *Motivations*

Les réseaux de neurones ont une propriété d'approximation universelle.

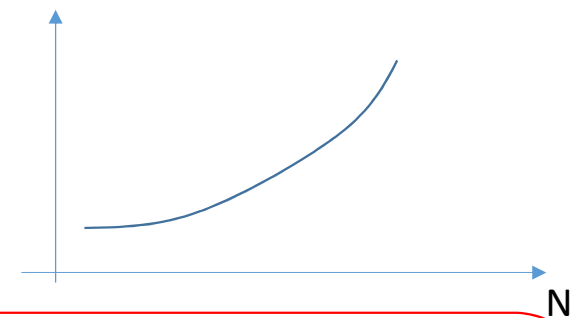
En règle générale, on montre qu'en ajoutant des neurones, l'erreur d'approximation (en apprentissage supervisé) **peut** tendre vers 0. Pour cela, il suffit d'ajouter des neurones dans les couches cachées.



Erreur d'approximation



Nombre de poids dans le réseau



On peut théoriquement réduire l'erreur d'approximation autant qu'on veut en augmentant  $N$

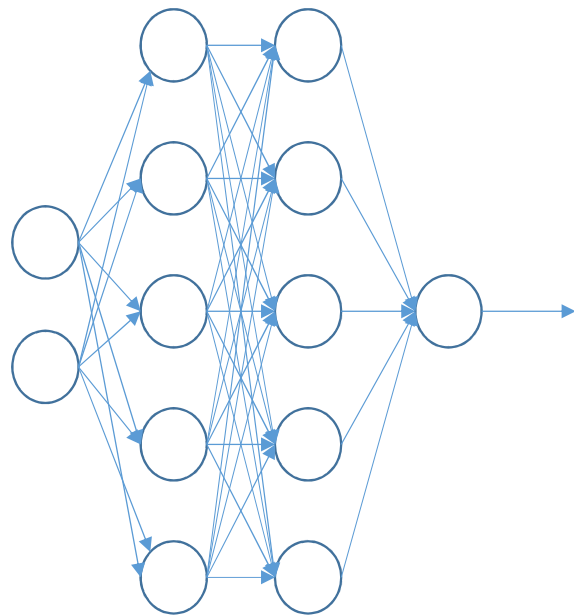
AU PRIX d'un nombre de poids de plus en plus important

PB: Les performances des algorithmes d'apprentissage des poids ne suivent pas

# Deep learning – *Motivations*

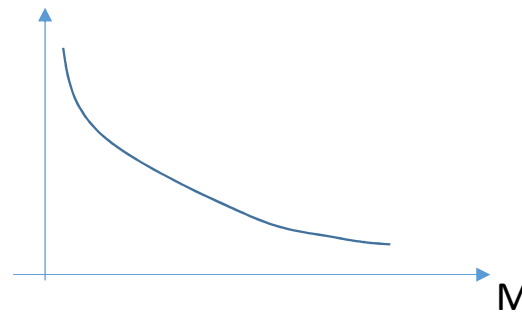
Les réseaux de neurones ont une propriété d'approximation universelle.

En règle générale, on montre qu'en ajoutant des neurones, l'erreur d'approximation (en apprentissage supervisé) **peut** tendre vers 0. Pour cela, il suffit d'ajouter des neurones dans les couches cachées.

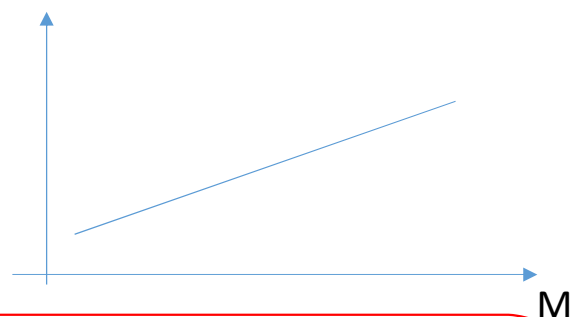


M neurones dans les couches cachées

Erreur d'approximation



Nombre de poids dans le réseau



En augmentant le nombre de couches cachées, on peut également diminuer l'erreur

ET

le nombre de poids dans le réseau augmente **linéairement**

=> Moins de difficultés avec l'algorithme d'apprentissage des poids

# Sommaire

- Quelques rappels basiques sur le traitement d'image
- L'algorithme du gradient stochastique
- Préparation des données d'entrées
  - Pourquoi des reseaux à convolution ?
  - **Les reseaux à convolution**
    - Construction et entrainement d'un reseau avec Matlab sur la base MNIST
    - Transfert learning
    - Visualisation des reseaux
      - Architecture de quelques reseaux

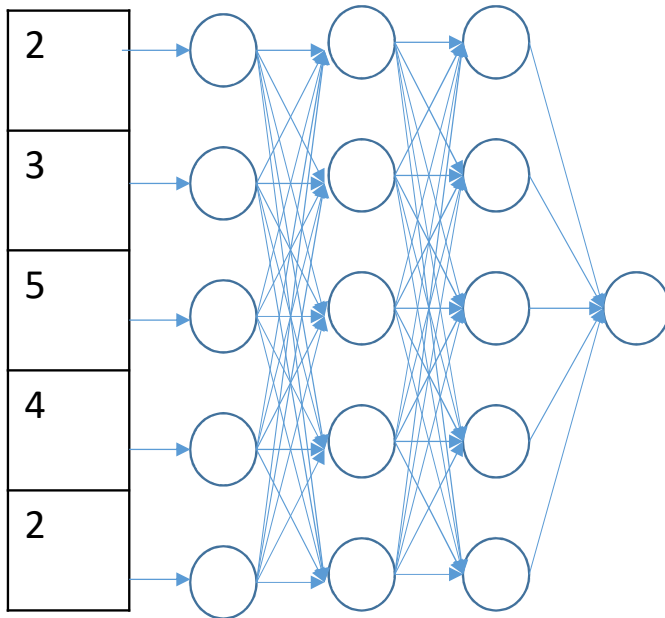


## Deep learning – Réseaux de neurones à convolution (conv net)

On souhaite traiter de très grandes quantités de données, comme la valeur des pixels d'une image

Problème : une petite image (200x200) en couleur nécessite 120 000 neurones dans la couche d'entrée.

Avec 2 couches complètement connectées, il faut  $2,88 \times 10^{10}$  poids  $\Rightarrow$  aucun algorithme d'apprentissage n'est capable de gérer cette quantité de poids.



Dans la pratique, on souhaiterait :

- traiter des images de taille plus conséquente
- Utiliser plus de couches (entre 10 et 20 par exemple)

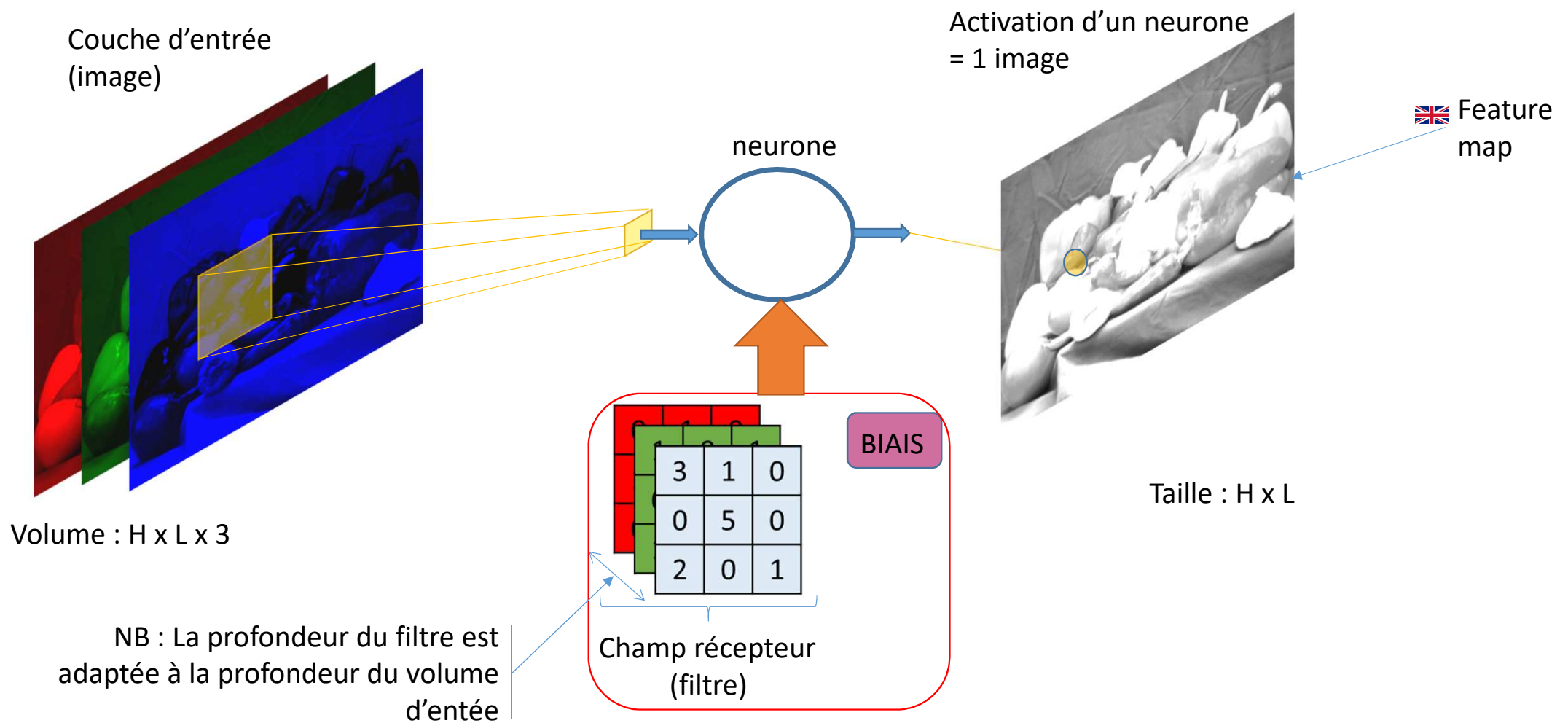
De plus, en « dépliant » l'image à l'entrée du réseau, on perd l'information sur la géométrie des pixels. Chaque pixel a des voisins. Une zone délimitée de l'image peut contenir des informations.

$\Rightarrow$  Il est peut probable que l'algorithme d'apprentissage des poids arrive à retrouver « par miracle » cette connectivité

$\Rightarrow$  Il faut faire autre chose

# Deep learning – Réseaux de neurones à convolution (conv net)

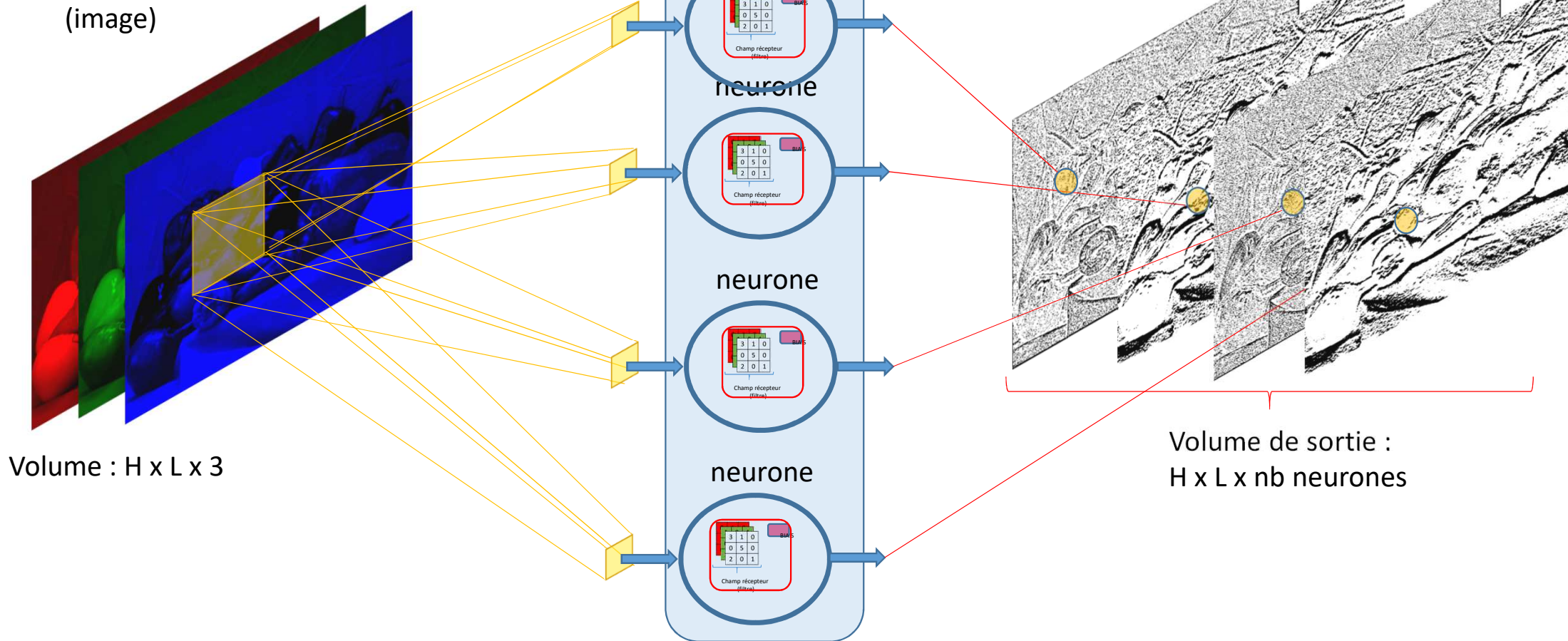
Pour limiter le nombre de poids, on va construire un neurone convolutif comme un filtre de taille réduite appliqué à toute l'image



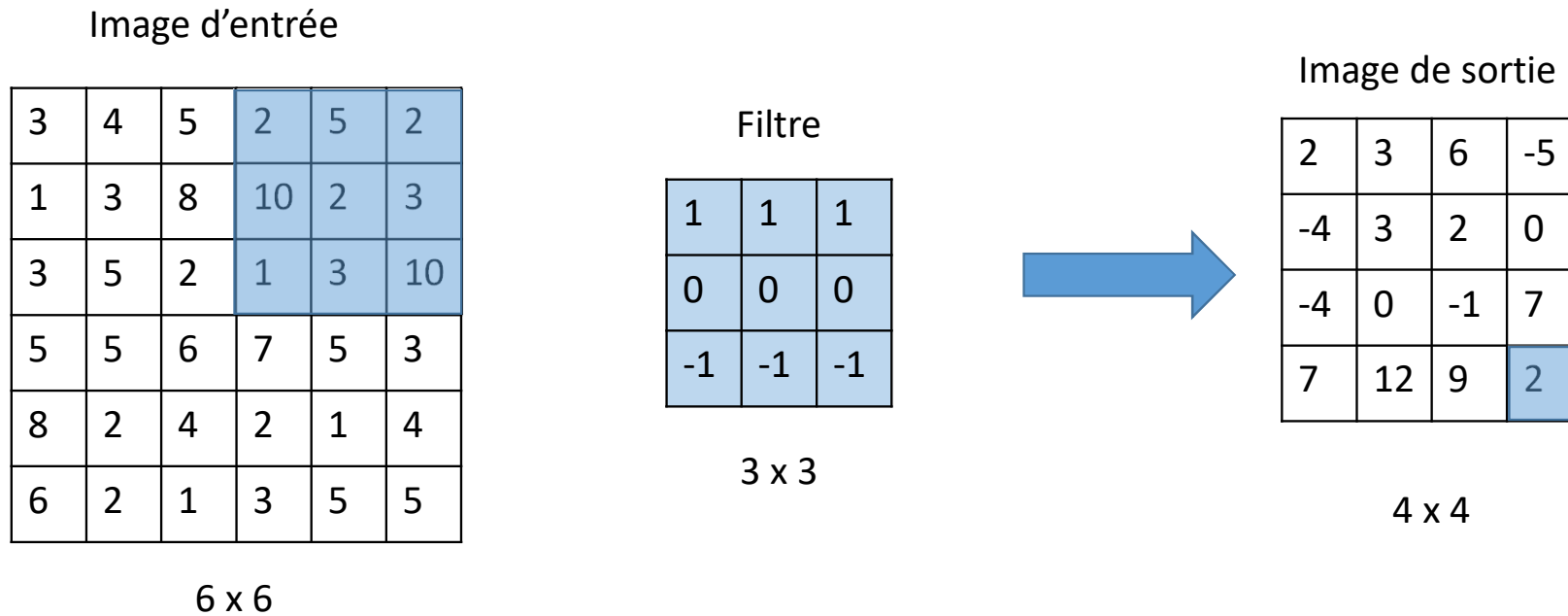
# Deep learning – Réseaux de neurones à convolution (conv net)

Pour limiter le nombre de poids, on va construire un neurone convolutif comme un filtre de taille réduite appliqué à toute l'image

Couche d'entrée  
(image)

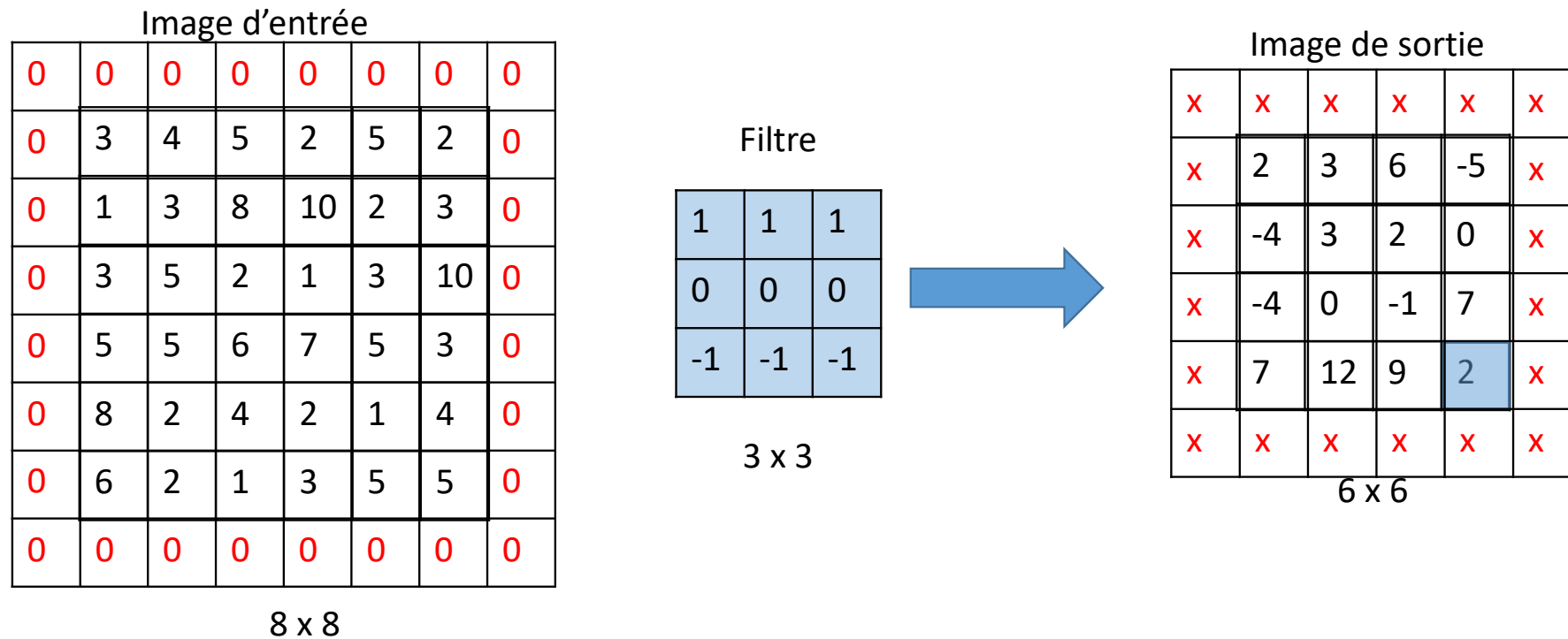


## Deep learning – *Gestion des bordures (padding)*



L'image de sortie est plus petite que l'image d'origine car on n'a pas appliqué le filtre en dehors de l'image d'origine.

## Deep learning – Gestion des bordures (padding)



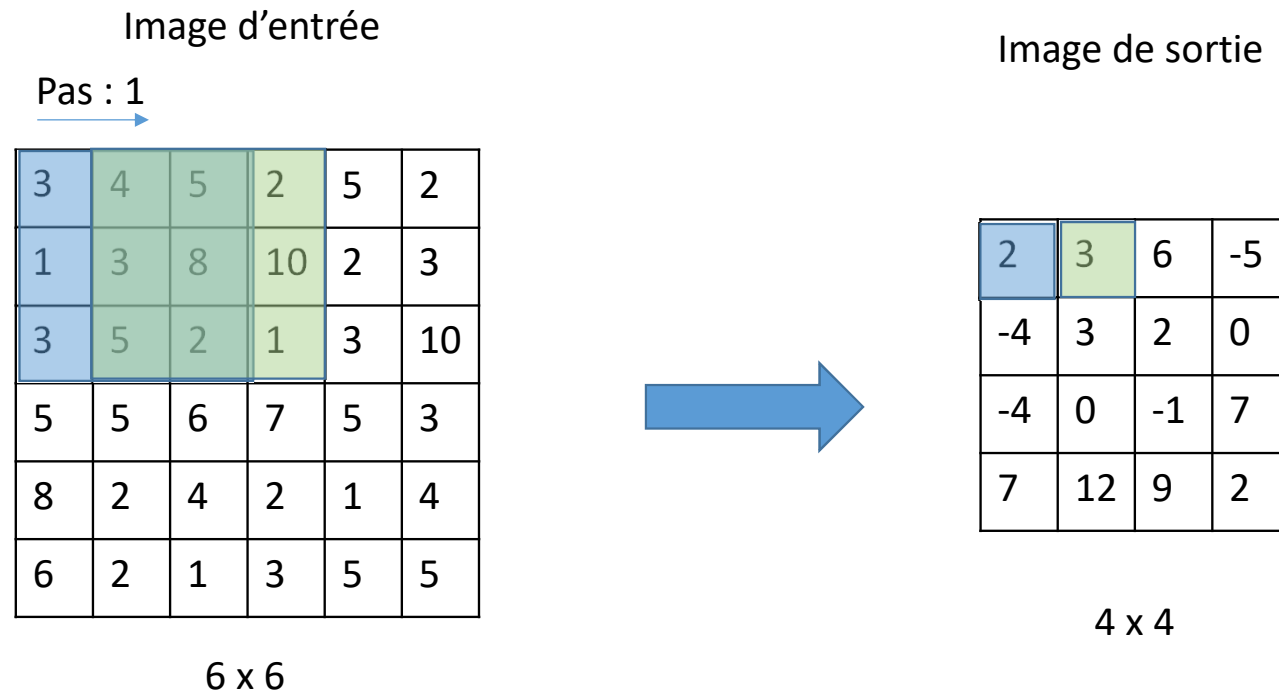
En rajoutant des zéros à l'extérieur, on peut augmenter la taille de l'image de sortie.

=> Le terme Anglais est « **zero Padding** » 🇬🇧

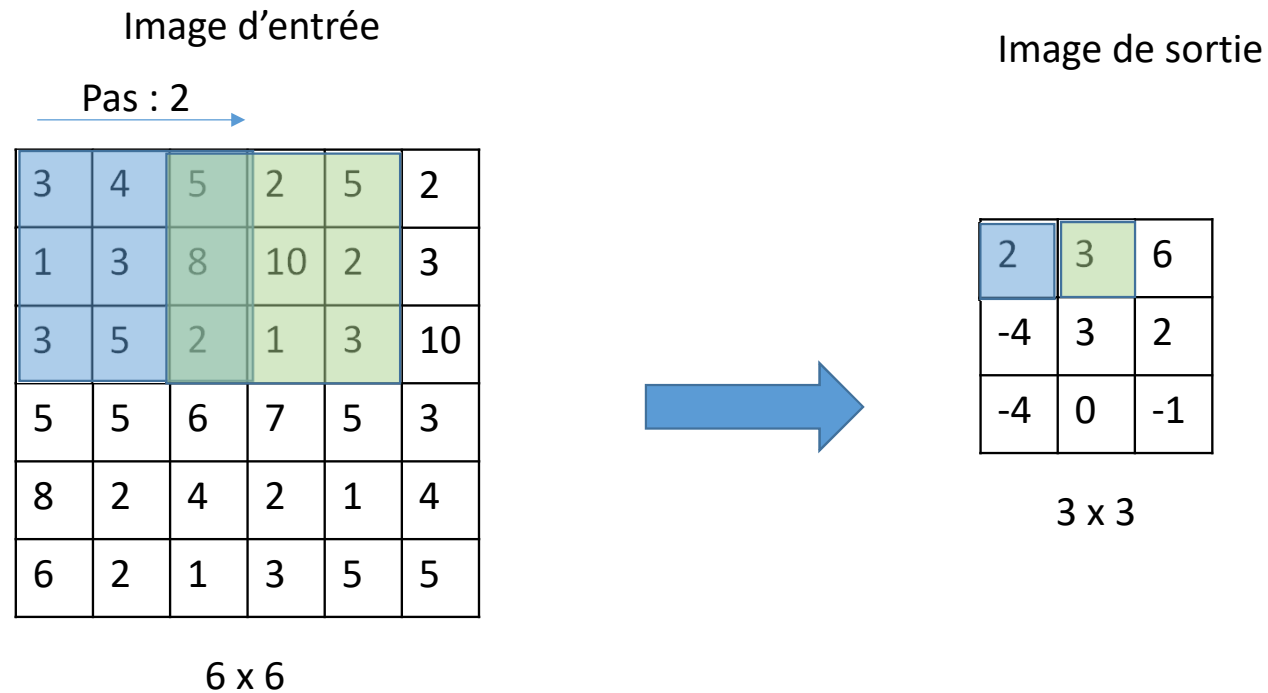
La plupart des logiciels permettent de calculer automatiquement le nombre de zéro à rajouter **pour préserver la taille de l'image d'entrée** (Padding='same' 🇬🇧 )



## Deep learning – *Gestion de l'avancée du filtre (stride)*



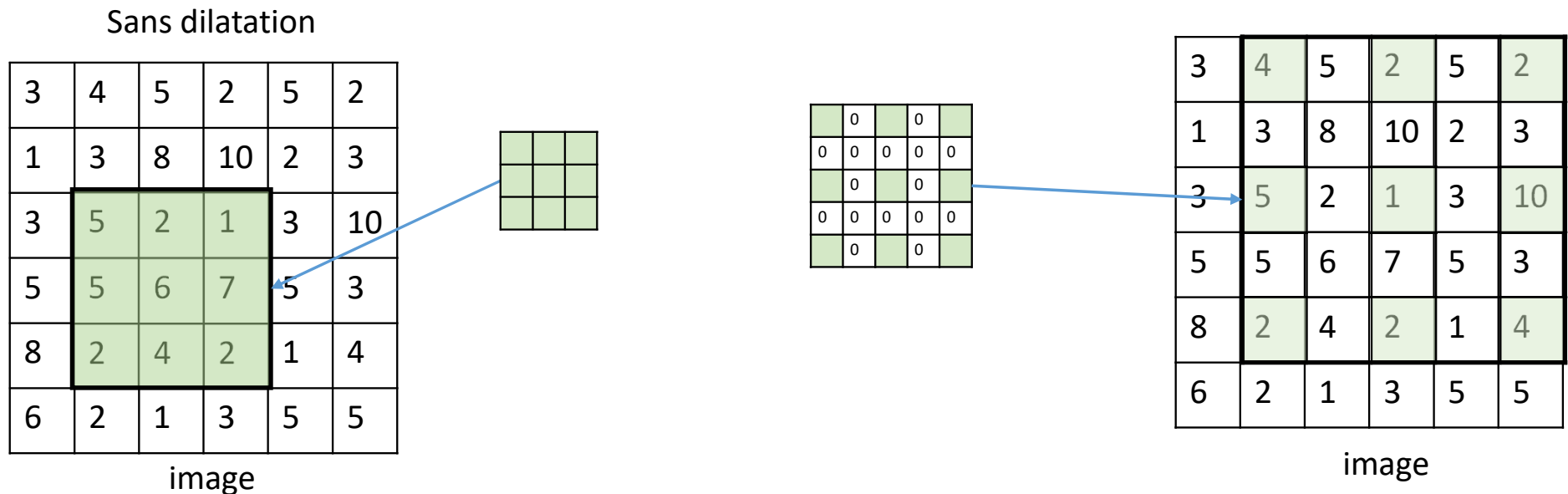
## Deep learning – Gestion de l'avancée du filtre (stride)



Le pas d'avancement du filtre ( 🇬🇧 stride) permet de diminuer la taille de l'image de sortie

## Deep learning – *Gestion de la dilatation*

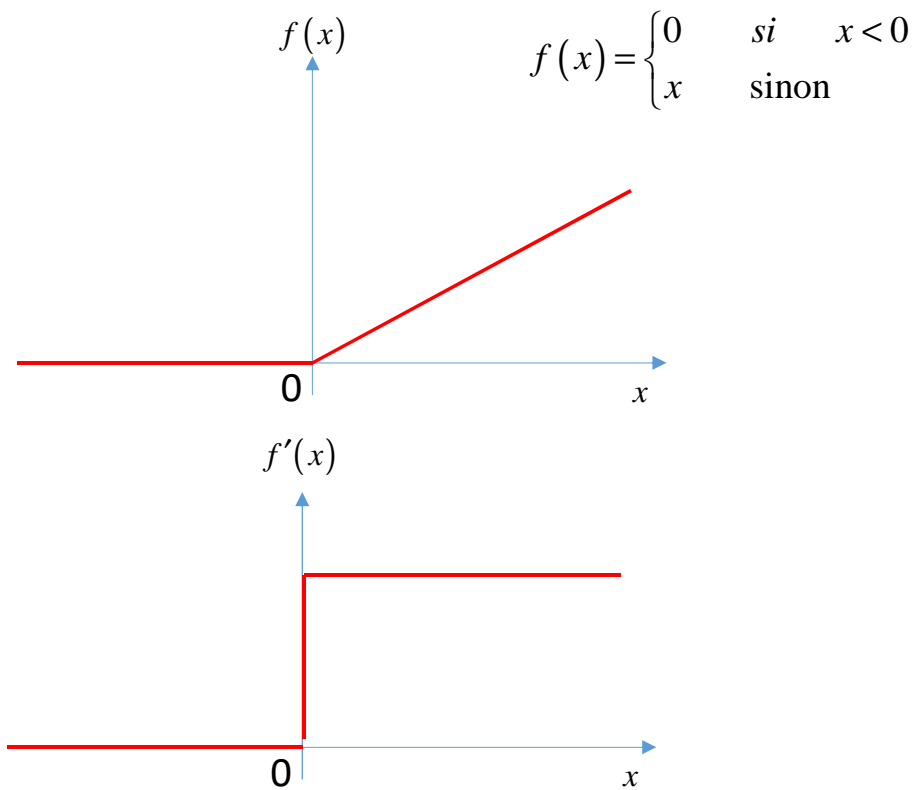
**La dilatation** permet d'obtenir un filtre plus grand sans ajouter de poids supplémentaire à apprendre. Pour cela, des zéros sont ajoutés entre les données du filtre




Résultat: Avec le même nombre de poids, le filtre «scanne » un plus grande zone de l'image MAIS la présence de zéros rends le filtre moins efficace qu'un filtre de taille équivalente sans dilatation.

## Deep learning – *Non linéarité ReLu*

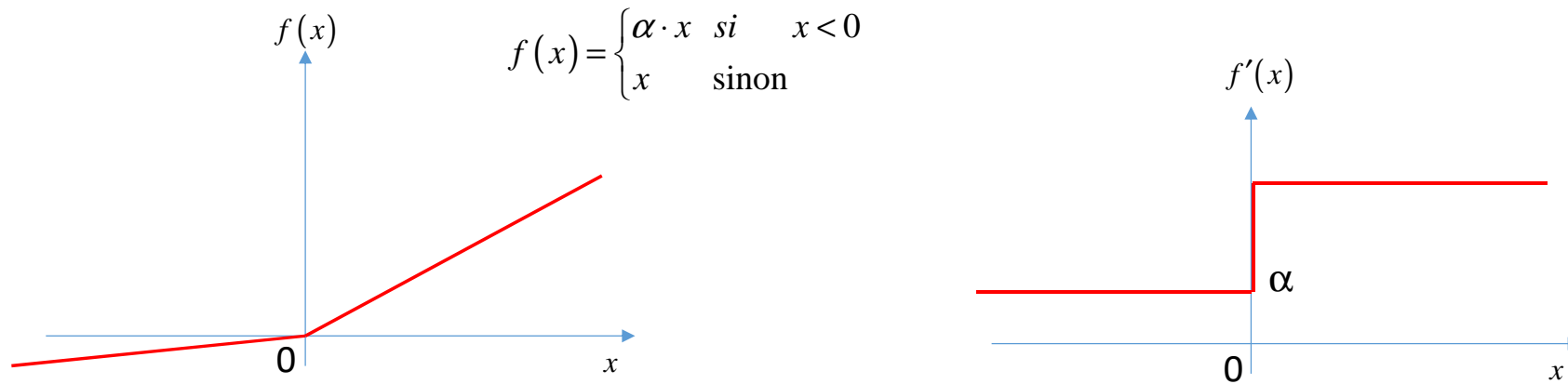
Une avancée significative dans les réseaux de neurones a été atteinte en utilisant une fonction d'activation particulière:  
REctifier Linear Unit : (Nair and Hinton, 2010)



Intérêt: le gradient est unitaire dans la zone positive:  
Il n'y a plus de problème de dilution du gradient  
( vanishing gradient)

## Deep learning – *Non linéarité Leaky - ReLu*

La non linéarité ReLu a un gradient nul pour les valeurs négatives. Cela peut poser des problèmes pour l'apprentissage. Une alternative est la non linéarité « Leaky ReLU »



Intérêt: permet d'avoir un gradient non nul pour un cout de calcul relativement faible

Inconvénient : la dérivée n'est pas continue

[1] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807-814. 2010.

## Deep learning – *Pooling*

Les couches de mise en commun (🇬🇧 pooling) permettent de réduire la quantité d'information à traiter.

L'idée de base est de considérer une zone de l'image de départ et d'en extraire un indicateur « pertinent » (max ou moyenne)

3	4	5	2	5	2
1	3	8	10	2	3
3	5	2	1	3	10
5	5	6	7	5	3
8	2	4	2	1	4
6	2	1	3	5	5


MaxPooling : le résultat est la plus grande valeur (10)

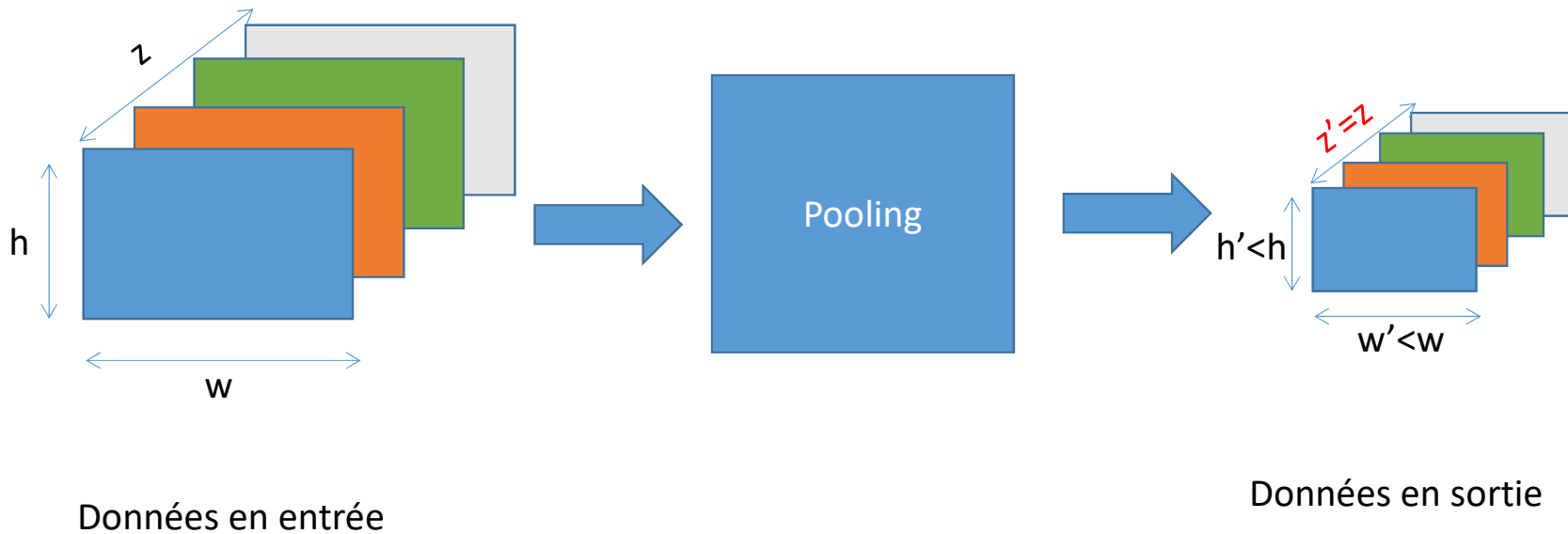
AvgPooling : le résultat est la moyenne de la zone (5.22)

# Deep learning – *Pooling*

L'opération est appliquée à chaque couche du volume d'entrée

=> Le volume de sortie à le même nombre de couches, mais ces couches sont plus petites

Le pas d'avancement du filtre (  stride) est en général égal à la taille du filtre afin de réduire la quantité d'information présente.



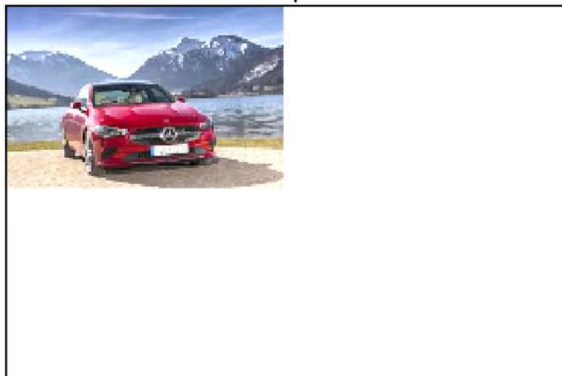
# Deep learning – *Pooling*

Exemple de résultats obtenus en appliquant un max pooling à une image.

image originale



max -  $n_{\text{pool}}=2$



max -  $n_{\text{pool}}=4$





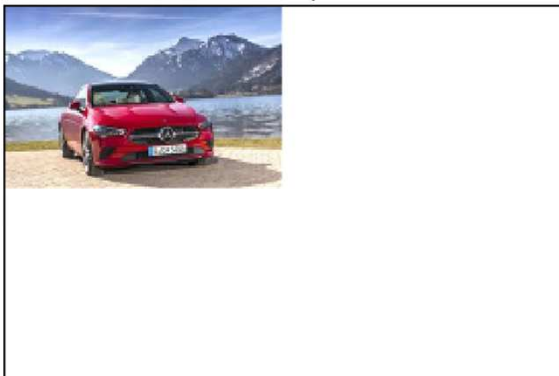
# Deep learning – *Pooling*

Exemple de résultats obtenus en appliquant un average pooling à une image.

image originale



average -  $n_{\text{pool}}=2$



average -  $n_{\text{pool}}=4$

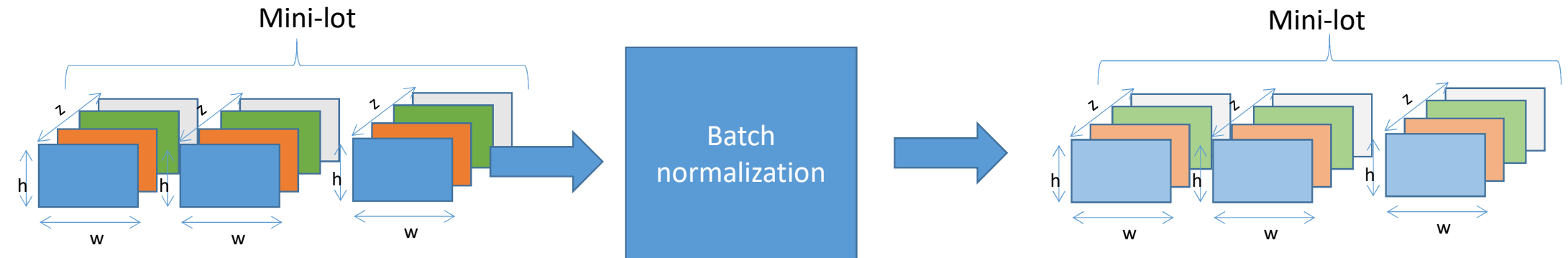


## Deep learning – *Batch normalization*

La couche de « batch normalisation » permet d'améliorer l'apprentissage par une régularisation des données circulant dans le réseau.

De la même manière que les données d'entrée ont été normalisées, il est important de normaliser les activations. Cela permet d'éviter les « explosions de gradients ».

La normalisation se fait sur un mini-lot.



BatchNormalization : **utilise 2 paramètres par tranche**, la normalisation se fait sur l'ensemble des données d'un mini-lot et pour plan du volume de sortie

BatchNormalization : utilise  $2 \times h \times w$  paramètres par tranche, la normalisation se fait sur l'ensemble des données d'un mini-lot et pour plan du volume de sortie

# Deep learning – *Batch normalization*

De la même manière que les données d'entrée ont été normalisées, il est important de normaliser les activations. Cela permet d'éviter les « explosions de gradients ».



L'algorithme consiste à :

- Normaliser les données (moyenne 0 et variance 1)
- Décaler le résultat précédent pour avoir une moyenne  $m$  et une variance  $v$  avec  $m$  et  $v$  2 paramètres devant être appris

BatchNormalization : utilise 2 paramètres par tranche (2z paramètres en tout), la normalisation se fait donc sur l'ensemble des données d'un mini-lot et pour chaque plan du volume de sortie

BatchNormalization2D : utilise 2 x w x h paramètres par tranche, la normalisation se fait sur l'ensemble des données d'un mini-lot et pour chaque « pixel » de chaque volume de sortie

Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *preprint, arXiv:1502.03167* (2015).

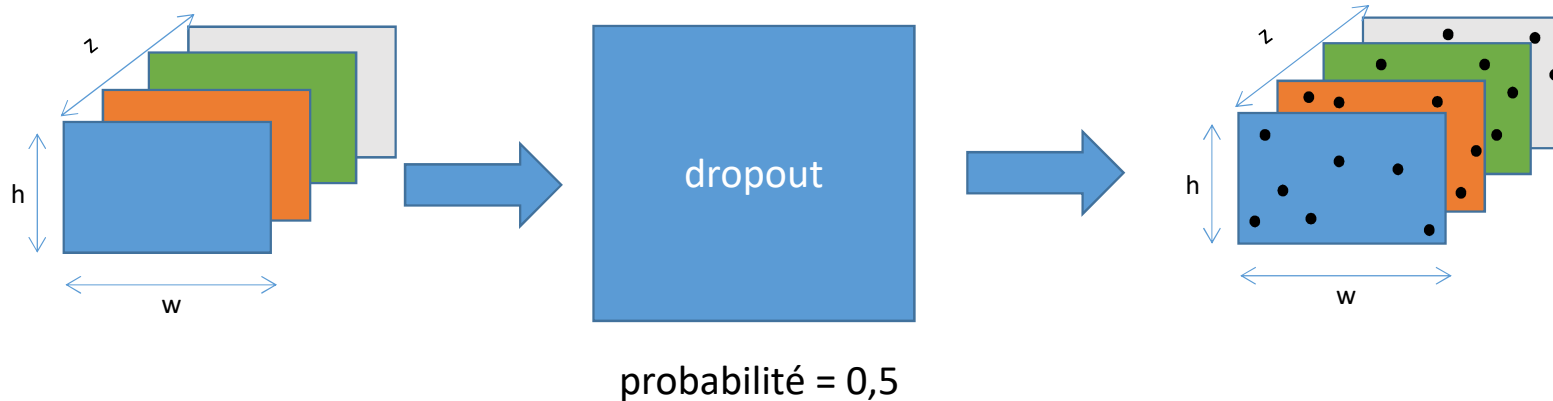
# Deep learning – *dropout*

/Srivastava & al. 2014/

- Structure du réseau: notion de « dropout »

Le « dropout » consiste à désactiver certaines activation du réseau. C'est une méthode de régularisation qui a pour but de faciliter l'apprentissage. Chaque activation est conservée avec une certaine probabilité.

Les auteurs suggèrent une probabilité de 0,5.



Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting , Journal of Machine Learning Research, 15(Jun):1929–1958, 2014.

## Deep learning – *couche finale*

/Srivastava & al. 2014/

- Fully connected

Pour les problèmes de classifications, des couches « denses » sont généralement rajoutée à la fin du réseau.

- softmax

A utiliser pour les problème de classification

- Regression layer

A utiliser avec Matlab pour les problème de régressions

# Sommaire

- Quelques rappels basiques sur le traitement d'image
- L'algorithme du gradient stochastique
- Préparation des données d'entrées
  - Pourquoi des reseaux à convolution ?
  - Les reseaux à convolution
    - Construction et entraînement d'un reseau avec Matlab sur la base MNIST
  - Transfert learning
  - Visualisation des réseaux
    - Architecture de quelques réseaux

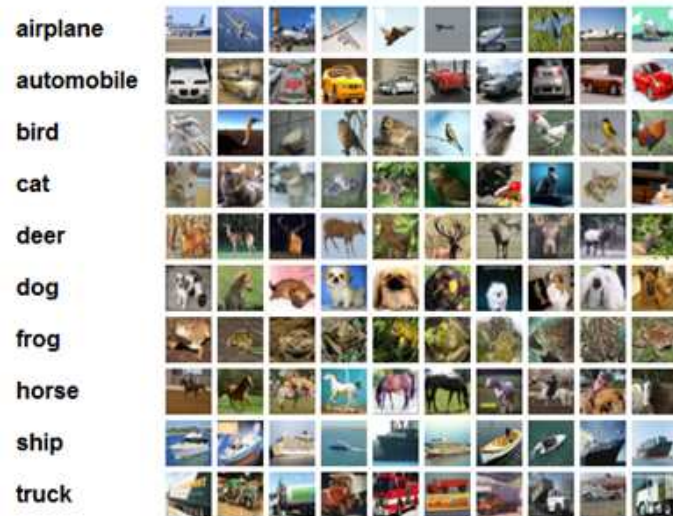


# Deep learning – *exemples*

Il existe de nombreux benchmark pour la classification:

- MNIST : classification de nombres écrit à la main – 60 000 images – meilleur résultat : 0,21% d'erreur sur la base de test /LeCun&al, 1998/
- CIFAR-10, 100 : 60 000 images, 10 classes – meilleur résultat: 3,4% d'erreur sur la base de test
- FACE CELEB A : 200 000 images de célébrités
- etc

CIFAR - 10



MNIST



# Deep learning – *programmation Matlab*

Classification des images de nombres de 0 à 9 (démonstration Matlab). Utilisation de la base MNIST (cf archive Moodle)

2 répertoires disponibles : 1 pour l'apprentissage et 1 pour le test

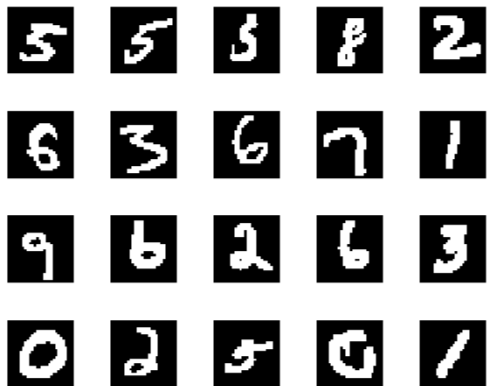
Nom du répertoire = classe des images

```
apprentissageData = imageDatastore([pwd '\MNIST\train'], 'IncludeSubfolders',true,'LabelSource','foldernames');  
nImages=length(validationData.Files);
```

`imageDatastore` : Objet permet le chargement automatique d'un grand nombre de données

⇒ Traite les images, les fichiers textes, les données csv, etc.

⇒ Permet de facilement découper un jeu de données en plusieurs lot (apprentissage, test, etc).



`apprentissageData.Files` : cell array avec le nom des fichiers

`Img=readimage(apprentissageData,No)` : lecture de l'image n° No

```
figure;  
perm = randperm(nImages,20);  
for i = 1:20  
    subplot(4,5,i);  
    imshow(apprentissageData.Files{perm(i)});  
end
```



# Deep learning – *programmation Matlab*

On peut vérifier le nombre d'images par catégories:

```
labelCount = countEachLabel(apprentissageData)
```

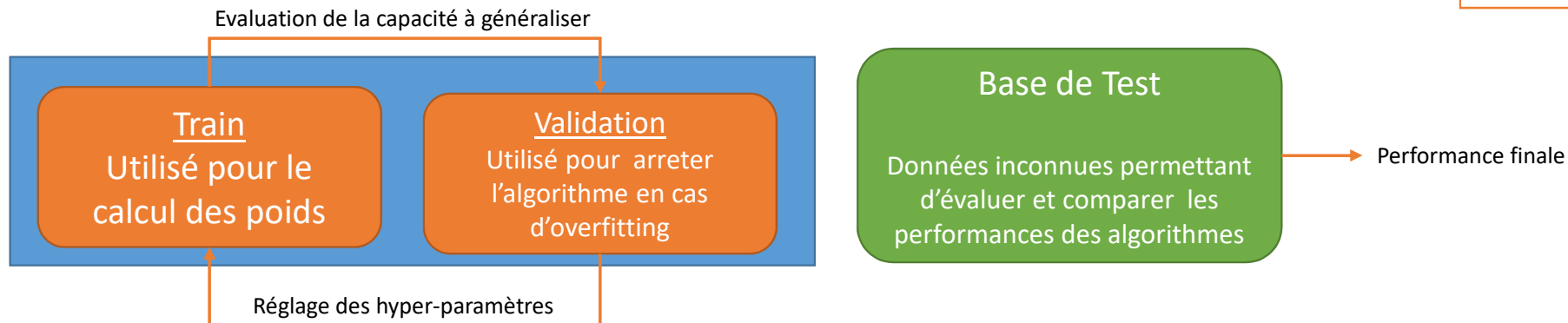
```
labelCount =  
10x2 table  
Label Count  
_____  
0      5923  
1      6742  
2      5958  
3      6131  
4      5842  
5      5421  
6      5918  
7      6265  
8      5851  
9      5949
```

Pour l'apprentissage des poids, on découpe la base de données d'apprentissage en 2 parties:

- Une grande partie des images sert à l'adaptation des poids
- Une partie plus faible sert à vérifier que le réseau généralise bien

```
trainNumFiles = ceil(min(labelCount.Count)*0.8);  
[trainDigitData,valDigitData] = splitEachLabel(apprentissageData,trainNumFiles,'randomize');
```

Base d'apprentissage



On vérifie la taille des images. Nécessaire pour spécifier la taille de la première couche du réseau

```
Img=readimage(apprentissageData,3);  
inputSize=size(Img);
```

```
inputSize =  
28 28
```

# Deep learning – *programmation Matlab*

Dans Matlab, un réseau de neurone se définit comme un tableau de couches.

Les principales couches disponibles sont:

- **imageInputLayer**([h w c]) => Images de hauteur h, largeur w, avec c=1 pour N&B et c=3 pour couleur
- **convolution2dLayer** (filterSize,numFilters)  
=> filterSize=[h w] Filtre de hauteur h, largeur w,  
=> numFilters : Nombre de filtres
- **batchNormalizationLayer** Normalisation des données:
  - 1) Les paramètres de normalisation initiaux (offset et gain) sont calculés sur un sous ensemble des données
  - 2) Les paramètres de normalisation sont ensuite adaptés comme les autres poids du réseau de neurones

⇒ Permet d'accélérer l'apprentissage

⇒ A utiliser systématiquement avant une couche ReLU
- **dropoutLayer(probability)** Couche de dropout avec spécification de la probabilité qu'un neurone soit désactivé

=> Il n'est pas nécessaire d'utiliser les couches de dropout avec les batchNormalizationLayer. Ces deux couches ont les même finalités.

## Deep learning – *programmation Matlab*

- **LeakyReLULayer**(alpha)
- **reluLayer**      Couche ReLU (pas de paramètres)
- **maxPooling2dLayer**(poolSize)
  - poolSize=[h w] : taille de la région de pooling
  - ⇒ Extrait la valeur maximale sur la région de taille h x w
  - ⇒ Permet de réduire la taille du réseau => moins de poids à apprendre dans les couches suivantes
- **fullyConnectedLayer**(outputSize)
  - ⇒ Couche entièrement connectée (classique)
  - ⇒ A n'utiliser qu'à la fin du réseau car nécessite l'apprentissage de beaucoup de poids
- **softmaxLayer**
  - ⇒ Couche *softmax* utilisée pour la classification
  - ⇒ Retourne une probabilité  $\in [0,1]$  pour chaque sortie

## Deep learning – *programmation Matlab*

- **RegressionOutputLayer**

⇒ Couche de régression permettant l'utilisation d'un critère Mean Squared Error:

⇒ Doit être précédé d'une couche « **fullyConnectedLayer** »

$$E = \sum_{i=1}^n \|y_i - \hat{y}_i\|^2$$

- **classificationLayer**

⇒ Couche permettant d'activer l'unique sortie correspondant à l'entrée avec la plus forte activation

⇒ Doit être précédé d'une couche « **softmaxLayer** »

⇒ Le critère utilisé pour l'apprentissage des poids est l'entropie croisée:

$$E = \sum_{i=1}^n \sum_{k=1}^{n_{classes}} y_i^k \ln(\hat{y}_i^k)$$

Valeur prédite par le réseau

Valeur exacte

## Deep learning – *programmation Matlab*

Pour chaque couche, il peut exister des paramètres additionnels:

- `Stride` : définit comment le filtre balaye l'image d'entrée
- `Padding` : définit si on rajoute des zéro à l'extérieur de l'image
- `WeightLearnRateFactor` : taux d'apprentissage des poids (multiple du taux global)
- `BiasLearnRateFactor` : taux d'apprentissage des biais (multiple du taux global)
- `WeightL2Factor` : coefficient de régularisation des poids (multiple du coef global)
- `BiasL2Factor` : coefficient de régularisation des biais (multiple du coef global)

Pour un problème de classification, les dernières couches du réseau seront toujours de la forme:

```
fullyConnectedLayer(nClasses)  
softmaxLayer  
classificationLayer
```

# Deep learning – *programming Matlab*

Définition du réseau de neurones:

```
layers = [imageInputLayer([inputSize 1])  
  
    convolution2dLayer(3,16,'Padding',1)  
    batchNormalizationLayer  
    reluLayer  
  
    maxPooling2dLayer(2,'Stride',2)  
  
    convolution2dLayer(3,32,'Padding',1)  
    batchNormalizationLayer  
    reluLayer  
  
    maxPooling2dLayer(2,'Stride',2)  
  
    convolution2dLayer(3,64,'Padding',1)  
    batchNormalizationLayer  
    reluLayer  
  
    fullyConnectedLayer(10)  
    softmaxLayer  
    classificationLayer];
```

[inputSize 1]=[28 28 1]

# Deep learning – *programming Matlab*

## Spécification des options d'apprentissage:

```
options = trainingOptions('sgdm',...  
    'MaxEpochs',3, ...  
    'ValidationData',valDigitData,...  
    'ValidationFrequency',30,...  
    'Verbose',false,...  
    'Plots','training-progress');
```

Un seul algo d'apprentissage disponible : sgdm



```
net = trainNetwork(trainDigitData, layers, options);
```

Apprentissage des poids (long)

```
save alldata
```

Sauvegarde du réseau entraîné (évite de refaire de longs calculs)

# Deep learning – *programming Matlab*

```
% Précision sur la base d'apprentissage
predictedLabels = classify(net,valDigitData);
valLabels = valDigitData.Labels;
accuracyValid = sum(predictedLabels == valLabels)/numel(valLabels)
```

Classifie les images de la base valData

```
% Vérification de la généralisation sur des images inconnues
testData = imageDatastore([pwd '\MNIST\valid'],...
    'IncludeSubfolders',true,'LabelSource','foldernames');

predictedTestLabels = classify(net,testData);
testLabels = testData.Labels;
accuracyTest = sum(predictedTestLabels == testLabels)/numel(testLabels)
```

Classifie les images de la base test

```
fprintf('Données de validation\n');
fprintf('  Précision : %.2f\n',accuracyValid);
fprintf('Données de test\n');
fprintf('  Précision : %.2f\n',accuracyTest);
```

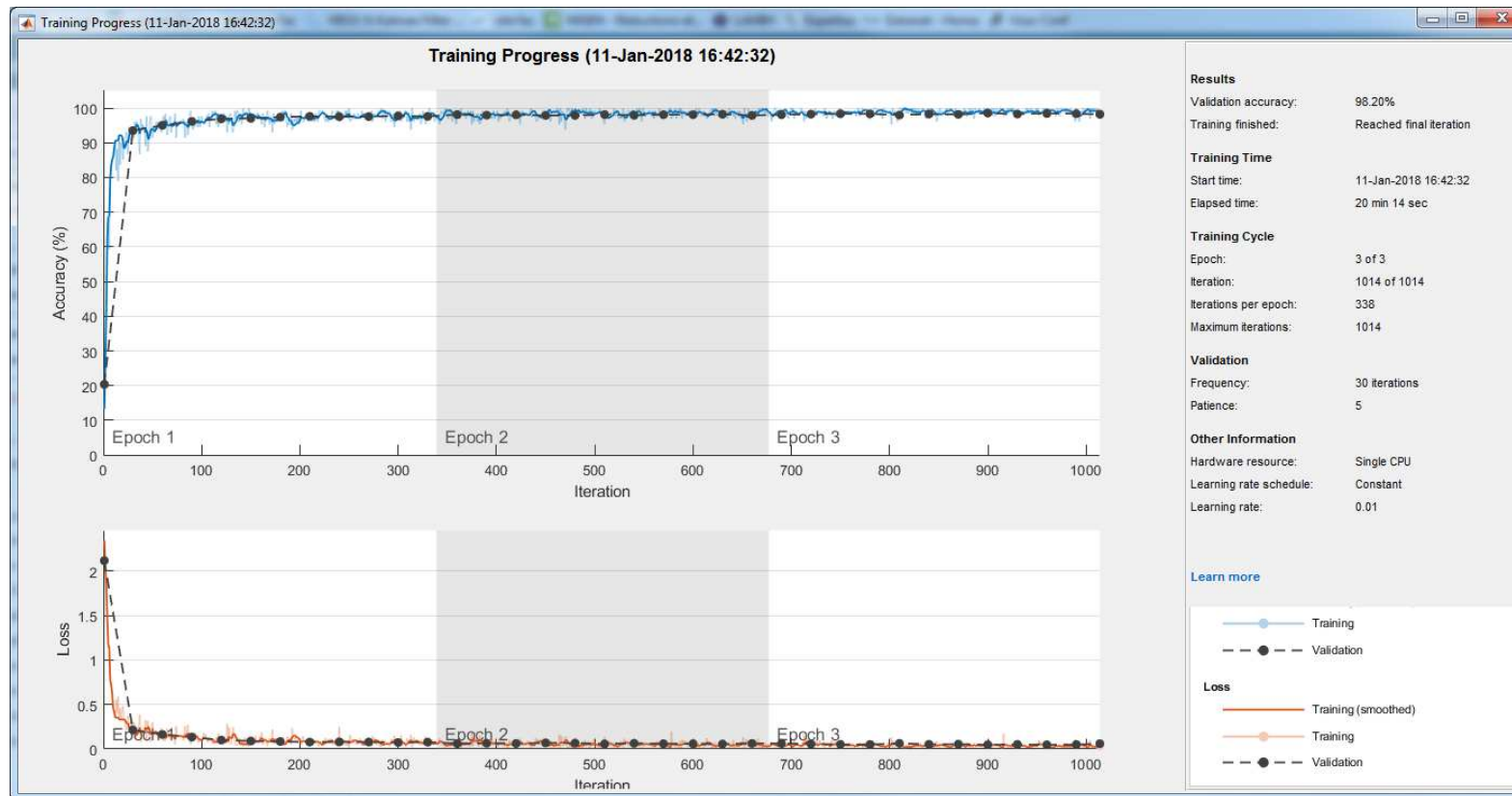
```
% Formatage des données
classeExacte=zeros(9,length(testLabels));
classePredite=zeros(9,length(testLabels));

for i=1:length(testLabels)
    classeExacte(testLabels(i),i)=1;
    classePredite(predictedTestLabels(i),i)=1;
end
plotconfusion(classeExacte,classePredite);
```



# Deep learning – *programming Matlab*

Classification des images de nombres MNIST – 20min d'apprentissage des poids



# Deep learning – *programming Matlab*

## Classification des images de nombres de 0 à 9 (d  mo Matlab)

Confusion Matrix										
Output Class	1	2	3	4	5	6	7	8	9	
	976 9.8%	0 0.0%	2 0.0%	0 0.0%	0 0.0%	3 0.0%	6 0.1%	0 0.0%	13 0.1%	0 0.0%
	0 0.0%	1133 11.3%	6 0.1%	0 0.0%	1 0.0%	0 0.0%	6 0.1%	2 0.0%	3 0.0%	7 0.1%
	2 0.0%	1 0.0%	1019 10.2%	4 0.0%	0 0.0%	0 0.0%	0 0.0%	3 0.0%	3 0.0%	0 0.0%
	0 0.0%	0 0.0%	0 0.0%	996 10.0%	0 0.0%	6 0.1%	0 0.0%	2 0.0%	1 0.0%	0 0.0%
	0 0.0%	0 0.0%	1 0.0%	0 0.0%	974 9.7%	0 0.0%	3 0.0%	1 0.0%	6 0.1%	10 0.1%
	0 0.0%	0 0.0%	0 0.0%	6 0.1%	0 0.0%	877 8.8%	3 0.0%	0 0.0%	1 0.0%	4 0.0%
	1 0.0%	1 0.0%	1 0.0%	0 0.0%	2 0.0%	2 0.0%	940 9.4%	0 0.0%	1 0.0%	1 0.0%
	1 0.0%	0 0.0%	3 0.0%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	1016 10.2%	5 0.1%	3 0.0%
	0 0.0%	0 0.0%	0 0.0%	2 0.0%	0 0.0%	1 0.0%	0 0.0%	1 0.0%	935 9.3%	0 0.0%
10	0 0.0%	0 0.0%	0 0.0%	0 0.0%	5 0.1%	3 0.0%	0 0.0%	3 0.0%	6 0.1%	984 9.8%
Target Class										
	1	2	3	4	5	6	7	8	9	10
	99.6% 0.4%	99.8% 0.2%	98.7% 1.3%	98.6% 1.4%	99.2% 0.8%	98.3% 1.7%	98.1% 1.9%	98.8% 1.2%	96.0% 4.0%	97.5% 2.5%

# Sommaire

- Quelques rappels basiques sur le traitement d'image
- L'algorithme du gradient stochastique
- Préparation des données d'entrées
  - Pourquoi des reseaux à convolution ?
  - Les reseaux à convolution
    - Construction et entrainement d'un reseau avec Matlab sur la base MNIST
    - **Transfert learning**
    - Visualisation des réseaux
      - Architecture de quelques réseaux



## Deep learning – *transfert learning*

Un des intérêt des réseaux de neurones est que s'ils ont été entraîné correctement, ils sont capable d'analyser les images d'entrée et en extraire de l'information de « haut niveau ».

L'entraînement d'un réseau de neurone est très couteux : il faut des cartes graphiques avec de gros processeur et beaucoup de mémore (12Go min), les plus chères dépassent les 10k€. Cela prends aussi beaucoup de temps : l'entraînement d'un réseau GoogleNet nécessite plus d'une semaine de calcul.

=>on peut réutiliser un réseau déjà entraîné :

- 1) Télécharger un réseau avec ses poids
- 2) Supprimer les dernières couches de classification (fully connected+softmax)
- 3) Rajouter un ensemble fully connected+softmax adapté au problème à résoudre.
- 4) Réentraîner le réseau avec de nouvelles données

Afin de ne pas « détruire » les poids déjà appris, pendant le nouvel entraînement, le taux d'apprentissage des couches transférées doit être très faible alors que celui des nouvelles couches doit être plus important.

Matlab 2017b propose les réseaux + poids suivants:  
AlexeNet, VGG16, VGG19, ResNet50, ResNet101, GoogLeNet,

Matlab 2019b permet également de télécharger des réseaux issus d'autres formats populaires.

# Deep learning – *transfert learning*

Exemple: entraînement d'un réseau AlexNet pour détecter des panneaux

On fait l'hypothèse qu'un réseau entraîné est capable d'extraire l'information « utile et structurée » des images.  
=> Il n'y a plus qu'à entraîner la dernière couche de classification



# Deep learning – *transfert learning*

## Exemple: entraînement d'un réseau AlexNet pour détecter des panneaux

```
clear all
close all
clc;

% Réseau de neurones initial : AlexNet
net=alexnet;
inputSize = net.Layers(1).InputSize(1:2)

% Chargement des images
fprintf('Chargement des images\n');
images = imageDatastore('Panneaux',...
    'IncludeSubfolders',true,...
    'LabelSource','foldernames');
images.ReadFcn = @(loc)imresize(imread(loc),inputSize);
fprintf('Fin chargement des images\n');

fprintf('Nombre d\'images total : %i\n',length(images.Files));

% Découpage des images en 2 jeux
[trainingImages,validationImages] = splitEachLabel(images,0.8,'randomized');
fprintf('Nombre d\'images pour l\'entraînement : %i\n',length(trainingImages.Files));
fprintf('Nombre d\'images pour la validation: %i\n',length(validationImages.Files));

% Affichage du nombre d'images
labelCount = countEachLabel(trainingImages)

numTrainImages = numel(trainingImages.Labels);
idx = randperm(numTrainImages,16);
figure
for i = 1:16
    subplot(4,4,i)
    I = readimage(trainingImages,idx(i));
    imshow(I)
end
```

# Deep learning – *transfert learning*

```
% Transfert learning with AlexNet
```

```
TransfertNet=net.Layers(1:22);
```

```
numClasses=length(categories(trainingImages.Labels));
```

```
fprintf('Nombre de classes : %i\n',numClasses);
```

```
NewNet=[TransfertNet      % Partie transférée
```

```
    fullyConnectedLayer(numClasses,'WeightLearnRateFactor',10,'BiasLearnRateFactor',10)
```

```
    softmaxLayer
```

```
    classificationLayer];
```

```
miniBatchSize = 10;
```

```
numIterationsPerEpoch = floor(numel(trainingImages.Labels)/miniBatchSize);
```

```
options = trainingOptions('sgdm',...
```

```
    'MiniBatchSize',miniBatchSize,...
```

```
    'MaxEpochs',4,...
```

```
    'InitialLearnRate',1e-4,...
```

```
    'Verbose',false,...
```

```
    'Plots','training-progress',...
```

```
    'ValidationData',validationImages,...
```

```
    'ValidationFrequency',numIterationsPerEpoch);
```

```
if 1==0
```

```
    netTransfer = trainNetwork(trainingImages,NewNet,options);
```

```
    save myNet netTransfer
```

```
else
```

```
    load myNet;
```

```
end
```

```
valLabels = validationImages.Labels;
```

```
predictedLabels = classify(netTransfer,validationImages);
```

On force un taux d'apprentissage important sur la nouvelle couche

On force un taux d'apprentissage faible pour les couches transférées

```
% Affichage
```

```
classeValidation=zeros(numClasses,length(valLabels));
```

```
classePrediction=zeros(numClasses,length(valLabels));
```

```
for i=1:length(valLabels)
```

```
    classeValidation(valLabels(i),i)=1;
```

```
    classePrediction(predictedLabels(i),i)=1;
```

```
end
```

```
plotconfusion(classeValidation,classePrediction);
```

# Deep learning – *transfert learning*

Résultat tous les panneaux sont détecté (0% d'erreur).

⇒ ATTENTION IL N'Y A QUE 28 IMAGES DANS LE JEU DE VALIDATION

⇒ CE N'EST PAS SUFFISANT POUR CONCLURE QUE LE RESEAU EST FONCTIONNEL

**Confusion Matrix**

Output Class	1	2	3	4	
	7 25.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
	0 0.0%	7 25.0%	0 0.0%	0 0.0%	100% 0.0%
	0 0.0%	0 0.0%	6 21.4%	0 0.0%	100% 0.0%
	0 0.0%	0 0.0%	0 0.0%	8 28.6%	100% 0.0%
	1	2	3	4	
Target Class					



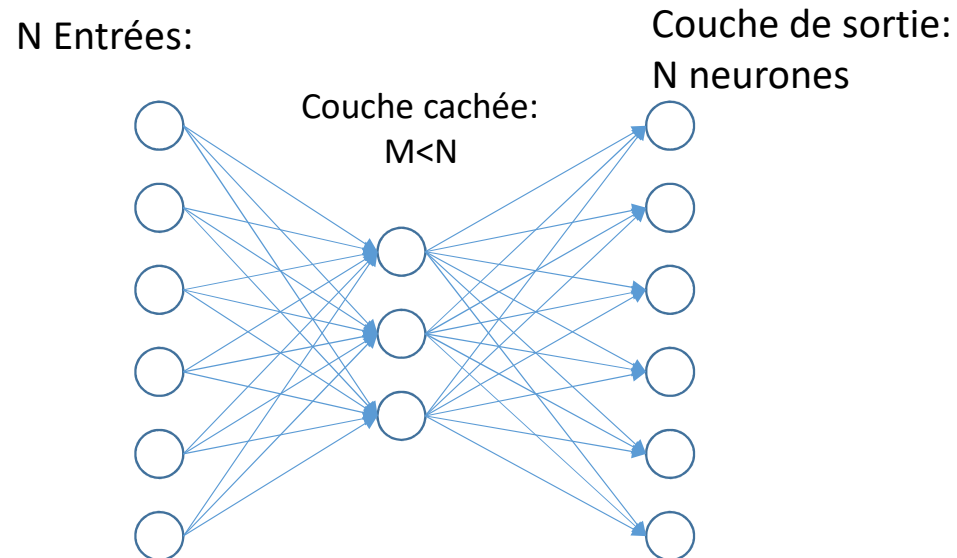
# Deep learning – *Autoencoders*

L'idée d'un Autoencoder est de forcer un réseau de neurone à apprendre la fonction identité:

=> Le réseau doit proposer à la sortie la même image que celle qui est fournie à l'entrée

Comme la couche cachée contient moins de neurones que de pixels à l'entrée, le réseau est obligé de compresser l'information:

- Ne peut fonctionner que si l'information dans les images d'entrée est structurée. Si à l'entrée on mets des images aléatoires, le réseau ne peut pas les reconstruire
- Il y a une perte d'information: l'image de sortie risque d'être « floue » : le réseau ne retient que l'essentiel



# Deep learning – *Autoencoders*

Avec Matlab, un AutoEncodeur est un réseau « classique » (i.e. pas à convolution).

Avantage: on peut utiliser un autoencodeur avec n'importe quel type de données (images, courbe, etc.)

Inconvénient: il faut ranger les images (toutes de même dimension) dans un cell array

```
clear all
close all
clc;

fprintf('Chargement des images\n');
apprentissageData = imageDatastore([pwd '\MNIST\train'],...
    'IncludeSubfolders',true,'LabelSource','foldernames');
fprintf('done\n');
nImages=length(apprentissageData.Files);
fprintf('%i images lues\n',nImages);

labelCount = countEachLabel(apprentissageData);

% Pour ne pas attendre trop longtemps (au détriment de la
% qualité) on
% diminue le nombre d'images
trainNumFiles = ceil(min(labelCount.Count)*0.1);
[trainDigitData,valDigitData] =
splitEachLabel(apprentissageData,trainNumFiles,'randomize');
nImages=length(trainDigitData.Files);
fprintf('%i images train\n',nImages);
```

# Deep learning – Autoencoders

```
% Affiche quelques images
figure;
perm = randperm(nImages,20);
for i = 1:20
    subplot(4,5,i);
    imshow(trainDigitData.Files{perm(i)});
end

% Convertie les images en cellarray
fprintf('Conversion cell array\n');
Imgs=cell(nImages,1);
for i=1:nImages
    Imgs{i}=readimage(trainDigitData,i);
end
fprintf('donne\n');
% Affiche les labels & le nombre d'images

Img=readimage(trainDigitData,3);
inputSize=size(Img);

hiddenSize1=100;
```

```
if 1==1
    autoenc1 = trainAutoencoder(Imgs,hiddenSize1, ...
        'MaxEpochs',400, ...
        'L2WeightRegularization',0.004, ...
        'SparsityRegularization',4, ...
        'SparsityProportion',0.15, ...
        'ScaleData', true);
    save DataAutoEncoder autoenc1
else
    load DataAutoEncoder
end
figure;
plotWeights(autoenc1);
```

# Sommaire

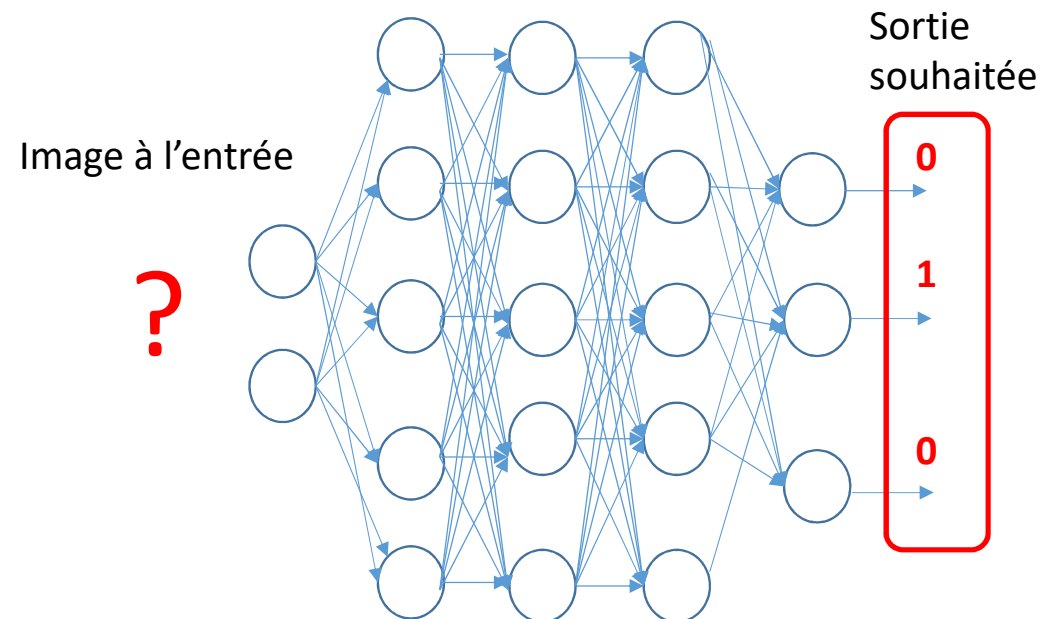
- Quelques rappels basiques sur le traitement d'image
- L'algorithme du gradient stochastique
- Préparation des données d'entrées
  - Pourquoi des reseaux à convolution ?
  - Les reseaux à convolution
    - Construction et entrainement d'un reseau avec Matlab sur la base MNIST
    - Transfert learning
      - Visualisation des reseaux
    - Architecture de quelques reseaux



## Deep learning – *visualisation*

Il est important de pouvoir « comprendre » ce que fait le réseau de neurones et l'algorithme d'apprentissage.

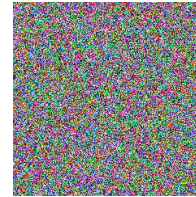
L'idée est d'utiliser le réseau (déjà entraîné) et un algorithme d'apprentissage pour calculer l'image à l'entrée du réseau qui maximise une sortie



# Deep learning – *visualisation (deep dreaming)*

Cf. <http://yosinski.com/deepvis> /Yosinski & al./ pour plus d'info.

Au début, on applique une image aléatoire à l'entrée du réseau:



Un algorithme optimise non pas les poids mais les valeur des pixels de l'image d'entrée

## Exemple:

On cherche l'image d'entrée qui maximise la classe « banjo »

Numéro de la couche (conv)

Numéro des classes

```
I = deepDreamImage(net, layer, channels, ...  
    'Verbose', true, ...  
    'NumIterations', iterations, ...  
    'PyramidLevels', levels);
```

Paramètres de l'algorithme  
+ grands = + de détail

# Deep learning – *visualisation (deep dreaming)*

```
clear all
close all
clc;

No=1;
switch No
    case 1
        net=alexnet;
        netname='AlexNet';
        layer = 23;
    case 2
        net=vgg16;
        netname='VGG16';
        layer = 39;
end
net.Layers
% for i=1:length(net.Layers(end).ClassNames)
%     fprintf('%i - %s\n',i,net.Layers(end).ClassNames{i});
% end
% return
levels = 2;
channels = [488]; % cellular telephone
iterations = 50;
I = deepDreamImage(net,layer,channels, ...
    'Verbose',true, ...
    'NumIterations',iterations, ...
    'PyramidLevels',levels);
Classname=net.Layers(end).ClassNames{channels};

name=sprintf('dreem_chl_%i_iter_%i_lvl_%i_%s',channels,iterations,levels,Classname);
imwrite(I,[name '.jpg']);
figure
imshow(I)
```

Affiche les couches: net.Layers(3) pour la 3<sup>ème</sup> couche

net.Layers(end).ClassNames: Liste des classes

## Deep learning – *visualisation (deep dreaming)*

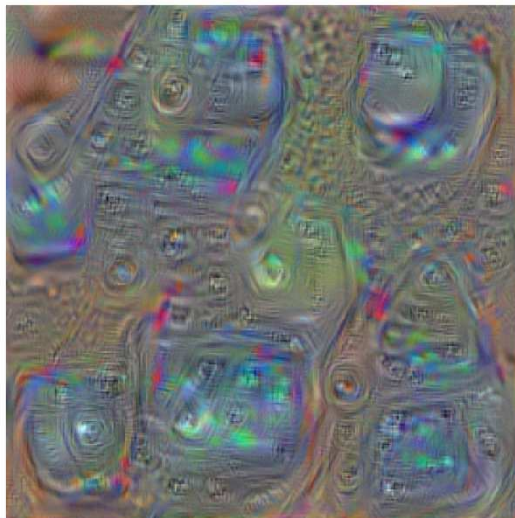
Etonnamment, les images générées ne ressemblent pas trop aux objets attendus.

⇒ Les images sont le résultat d'une optimisation

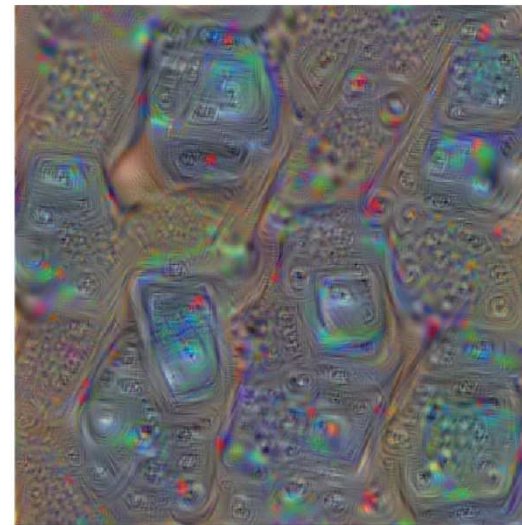
⇒ Si l'apprentissage était parfait, l'image générée serait en quelque sorte l'image moyenne permettant d'activer une classe

Le 'fantome' du téléphone apparaît à plusieurs endroits peut être par ce que le réseau est capable de détecter un téléphone à plusieurs endroits dans l'image

AlexNet – Classe 488 (cellular telephone)  
50 iterations – level 2



AlexNet – Classe 488 (cellular telephone)  
50 iterations – level 3





## Deep learning – *visualisation (deep dreaming)*

De nombreux chercheurs ont poussé l'idée plus loin.

Si on applique une image à l'entrée qui n'est pas aléatoire, alors le problème d'optimisation est le suivant:  
« trouver l'image (proche de celle fournie) à l'entrée qui active le plus une classe donnée »

=> On peut fournir un poisson à l'entrée et chercher à activer la classe golden retriever...



Class=« thunder snake »



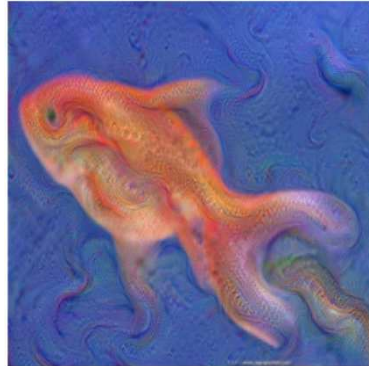
Class = 53  
Lvl = 2  
Iter = 10

## Deep learning – *visualisation (deep dreaming)*

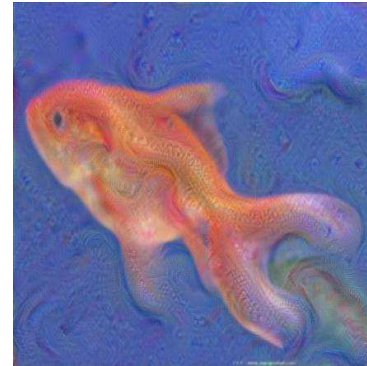
AlexNet



Lvl = 1 - Iter = 10



Lvl = 1 - Iter = 100

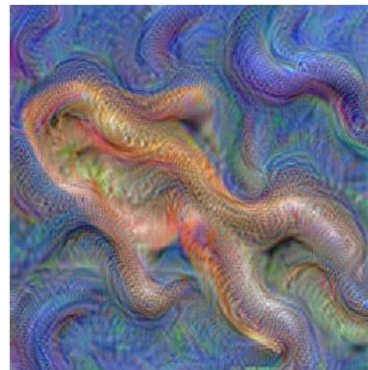


Lvl = 3 - Iter = 10

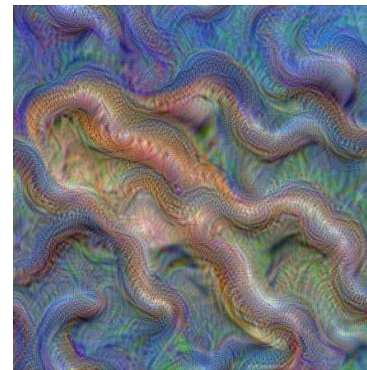
AlexNet



Lvl = 2 - Iter = 10



Lvl = 2 - Iter = 100



Lvl = 3 - Iter = 100

## Deep learning – *visualisation (activation)*

Au-delà des visualisations ‘récréatives’ précédentes, il est possible de visualiser les activations des différentes couches du réseau. On peut ainsi (tenter de) comprendre à quels stimuli répondent les filtres composant les différentes couches.



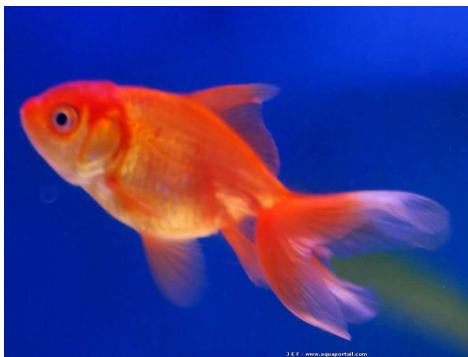
Sortie des 96 filtres de la 1ere couche d'un AlexNet



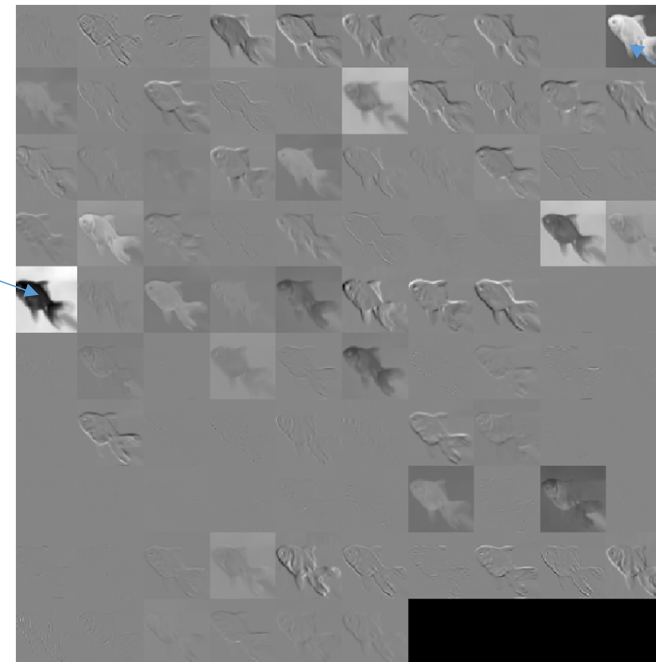
## Deep learning – *visualisation (activation)*

Au-delà des visualisations 'récréatives' précédentes, il est possible de visualiser les activations des différentes couches du réseau. On peut ainsi (tenter de) comprendre à quels stimuli répondent les filtres composant les différentes couches.

Sortie des 96 filtres de la 1ere couche d'un AlexNet



Réagit au bleu



Réagit au rouge

# Deep learning – *visualisation (activation)*

Dans les couches les plus faibles, les filtres réagissent à des caractéristiques « simples » (couleur, gradient, etc). Plus on remonte dans le réseau et plus les features sont de haut niveau



```
clear all  
close all  
clc
```

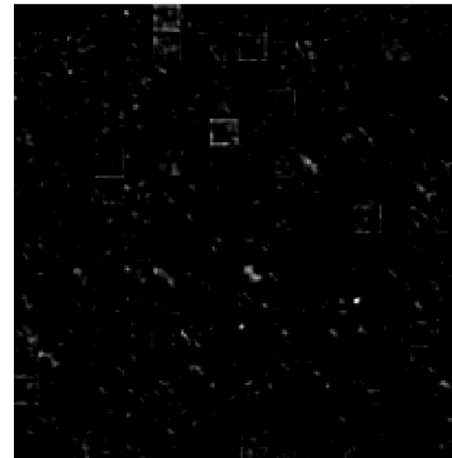
```
net=alexnet;  
inputSize=net.Layers(1).InputSize;
```

```
img=imread('Poisson.jpg');  
img=imresize(img,inputSize(1:2));  
act1 = activations(net,img,'relu5','OutputAs','channels');
```

```
% Redimensionnement  
sz = size(act1);  
act1 = reshape(act1,[sz(1) sz(2) 1 sz(3)]);  
montage(mat2gray(act1))
```

Nom ou n° de la couche

Sortie des de la couche ReLu5 d'AlexNet



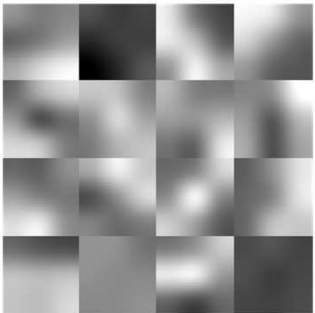
## Deep learning – *visualisation (poids)*

Dans l'approche deep learning, l'hypothèse est qu'un réseau bien entraîné va structurer les poids et les biais de telle sorte que les activations de chaque couche représentent une caractéristique des images appliquées à l'entrée du réseau: détection de contour, de tache, etc.

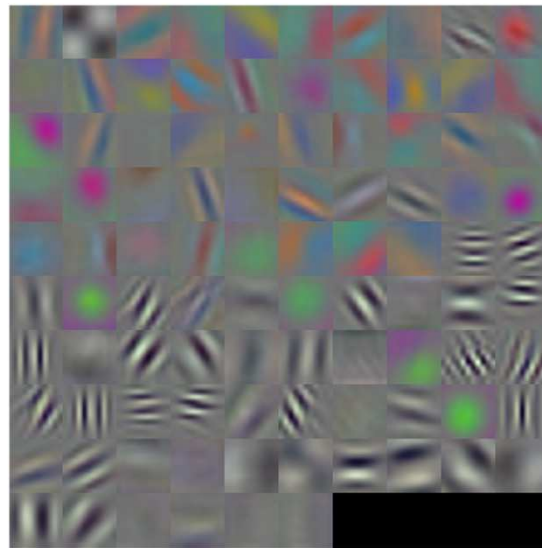
En général, lorsque le réseau a des poids structurés correctement, le réseau présente de bonne capacité de généralisation.

Réseau entraîné pour classifier des  
nombres manuscrits

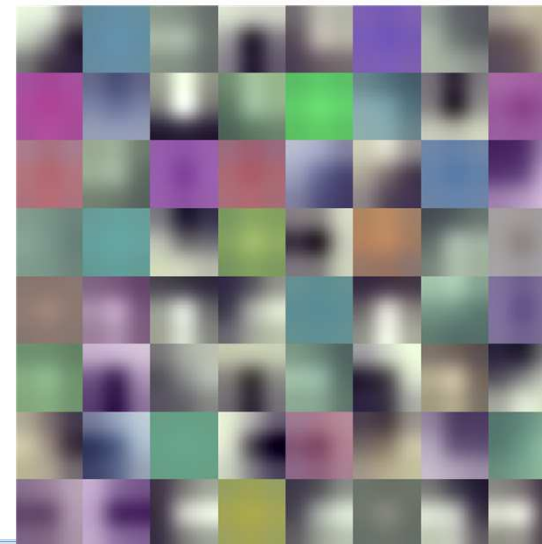
First convolutional layer weights - classique



AlexNet



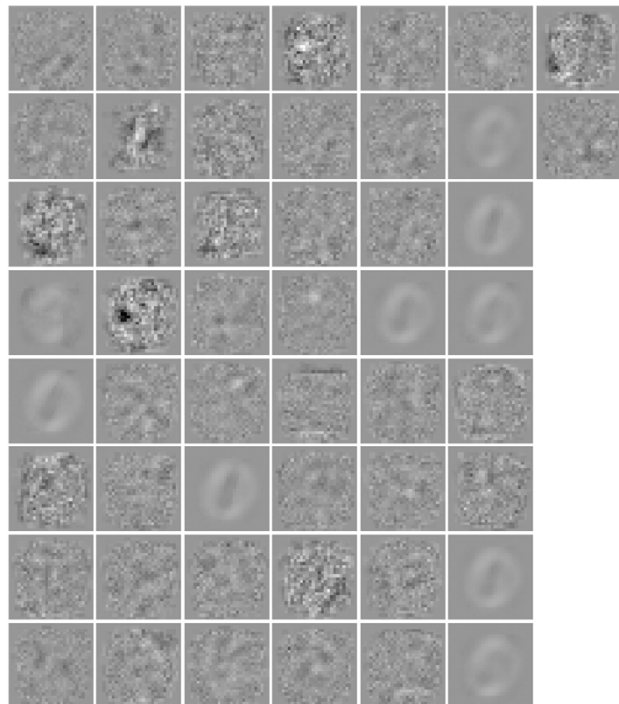
VGG16



## Deep learning – *visualisation (poids)*

Dans l'approche deep learning, l'hypothèse est qu'un réseau bien entraîné va structurer les poids et les biais de telle sorte que les activations de chaque couche représentent une caractéristique des images appliquées à l'entrée du réseau: détection de contour, de tache, etc.

En général, lorsque le réseau a des poids structurés correctement, le réseau présente de bonne capacité de généralisation.



Ce réseau est probablement mal entraîné : certains poids ressemblent à du bruit



# Sommaire

- Quelques rappels basiques sur le traitement d'image
- L'algorithme du gradient stochastique
- Préparation des données d'entrées
  - Pourquoi des reseaux à convolution ?
  - Les reseaux à convolution
    - Construction et entrainement d'un reseau avec Matlab sur la base MNIST
    - Transfert learning
    - Visualisation des reseaux
    - Architecture de quelques reseaux

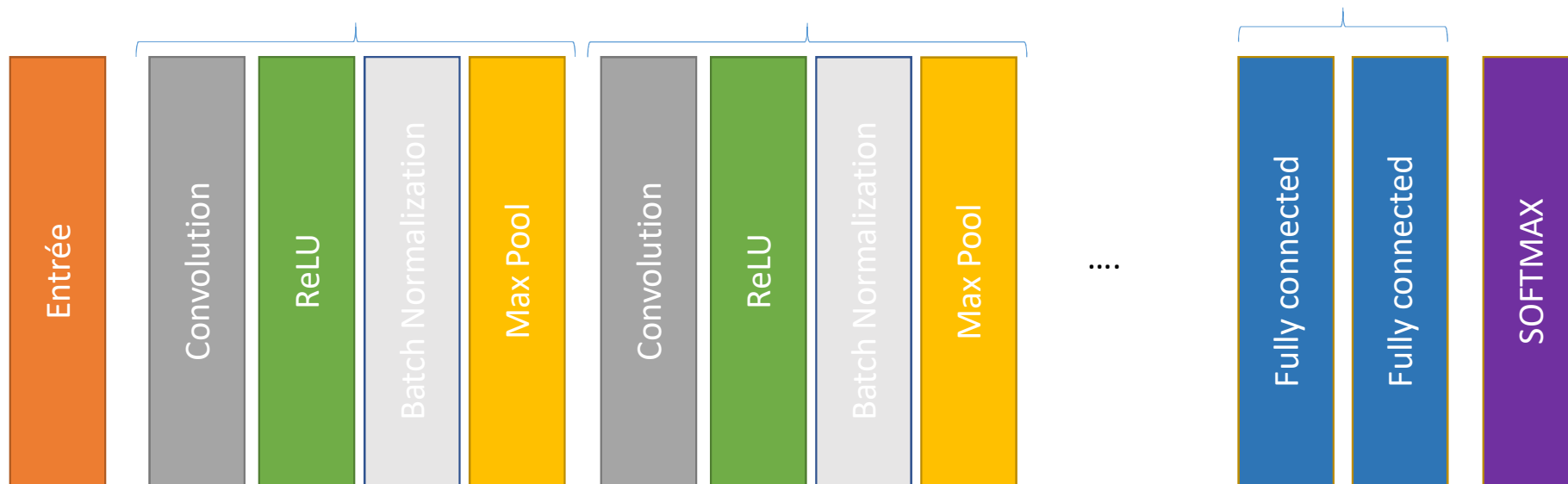




## Deep learning – *structure d'un réseau profond*

Il n'existe pas de structure idéale. Il faut adapter la structure du réseau à chaque application et aux spécificités des données manipulées.

En général, la structure optimale du réseau est inconnue, mais un réseau est généralement constitué par une succession de motifs élémentaires et terminé par une des couches denses (🇬🇧 fully connected).



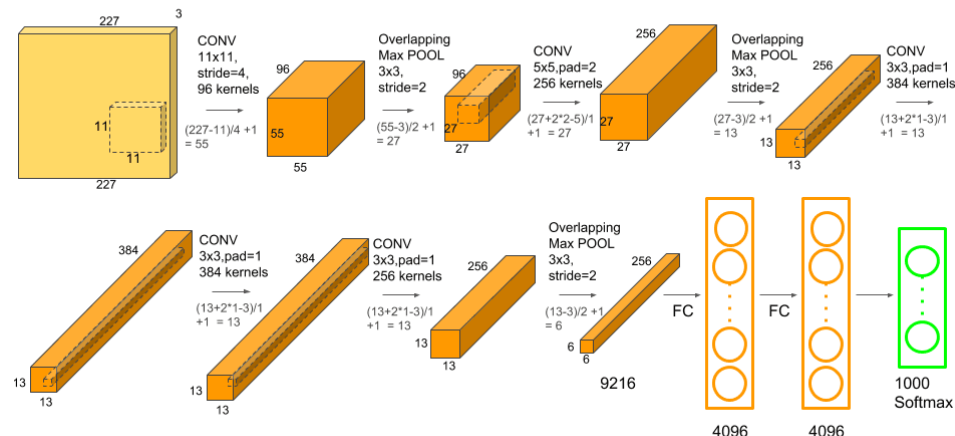
# Deep learning – *structure d'un réseau profond*

La structure d'un réseau et les hyper-paramètres associés sont choisis par le concepteur. Ces choix sont cruciaux peuvent changer les performances et la difficulté d'apprentissage du tout au tout.

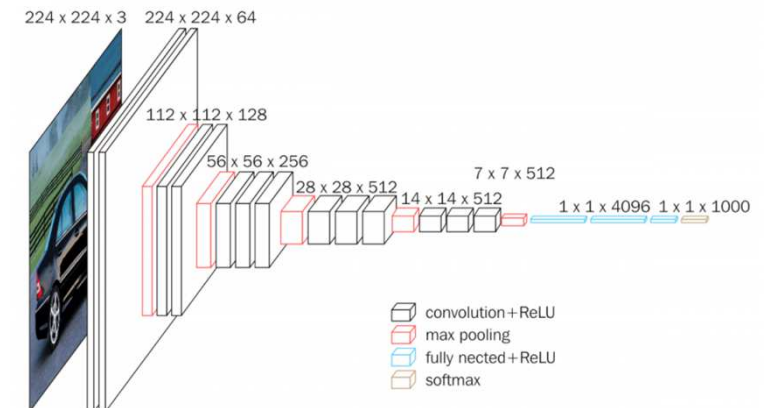
Il existe des benchmarks qui permettent de mettre en évidence les performances de classifications des réseaux. Certains réseaux sont réputés pour leur performances. Leur structure est en général connue et pour quelques réseaux les poids sont même mis à disposition de la communauté scientifique.

=> Par exemple : [http://caffe.berkeleyvision.org/model\\_zoo.html](http://caffe.berkeleyvision.org/model_zoo.html)

Alex net

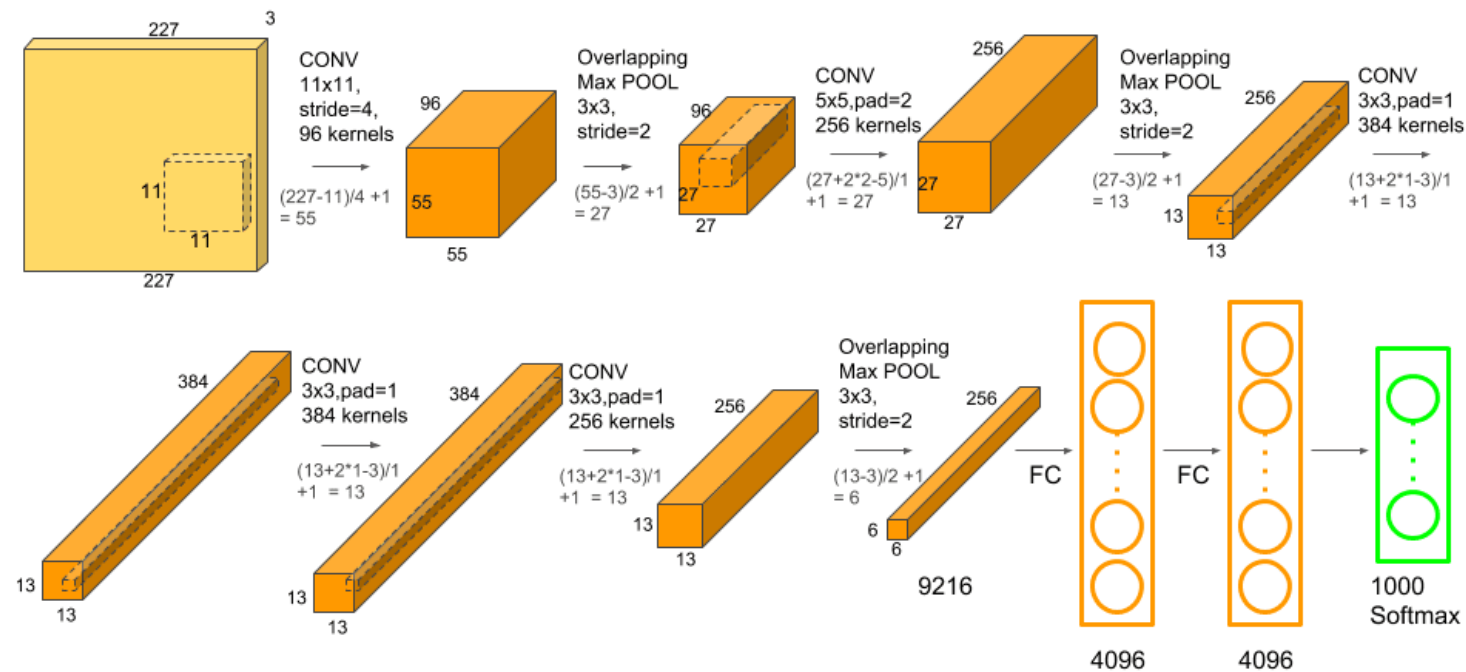


VGG 16




# Deep learning – Alexnet

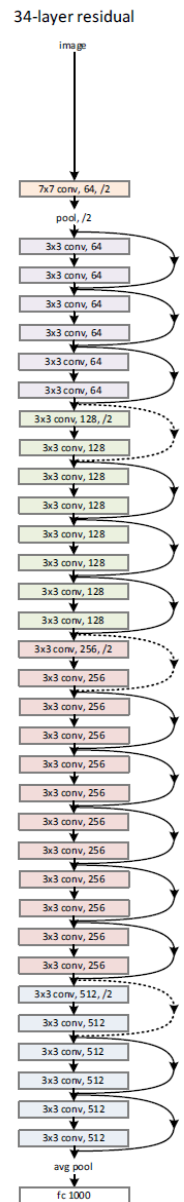
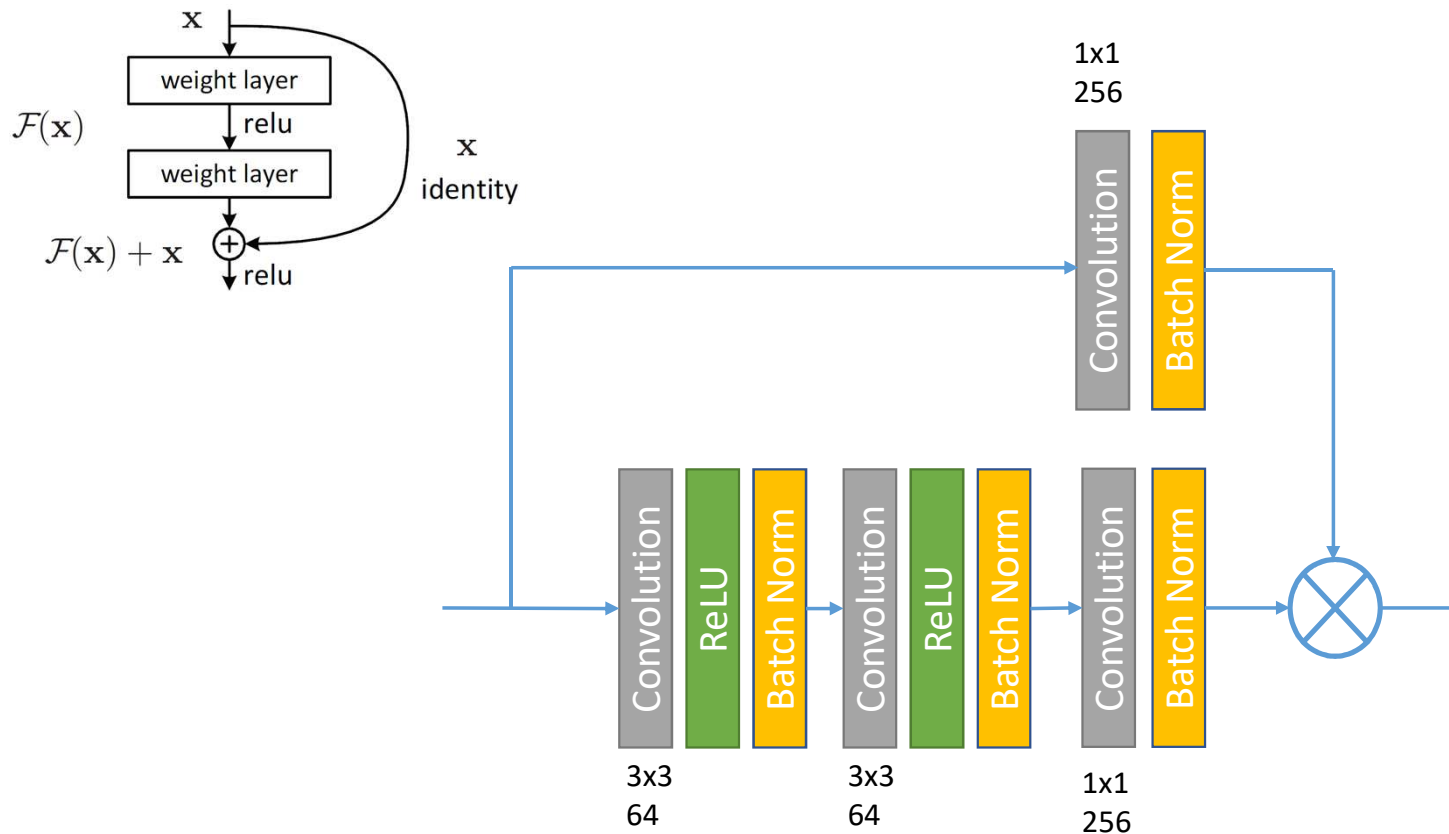
AlexNet est un des premiers réseau profond qui a réalisé une amélioration significative des performance sur le benchmark 201



Alex Krizhevskyn Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, January 2012  
Advances in neural information processing systems 25(2), DOI: [10.1145/3065386](https://doi.org/10.1145/3065386)

# Deep learning – ResNet

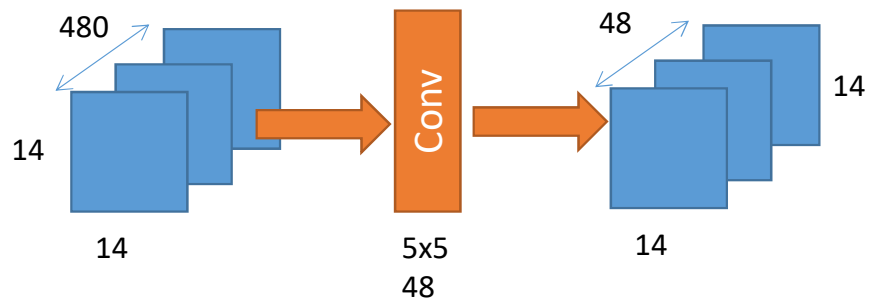
En 2013, Microsoft propose le réseau ResNet qui introduit la notion de « réseau à Residu » (  residual network)



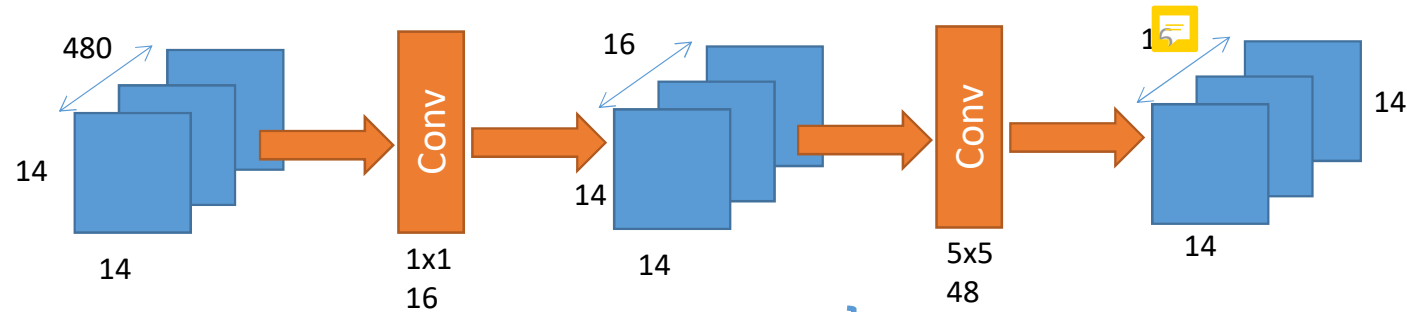
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, **Deep Residual Learning for Image Recognition**, 2015, <https://arxiv.org/abs/1512.03385>

## Deep learning – *GoogleNet (inception v1)*

En 2014, Google propose un mécanisme permettant de diminuer le nombre d'opérations via l'utilisation de convolutions 1x1



Nb operations:  $14 \times 14 \times 48 \times 5 \times 5 \times 480 = 112,9 \text{ Millions}$



Nb operations1:  $14 \times 14 \times 16 \times 1 \times 1 \times 480 = 1,5 \text{ Millions}$

Nb operations2:  $14 \times 14 \times 48 \times 5 \times 5 \times 16 = 3,8 \text{ Millions}$

Total : 5,3 Millions

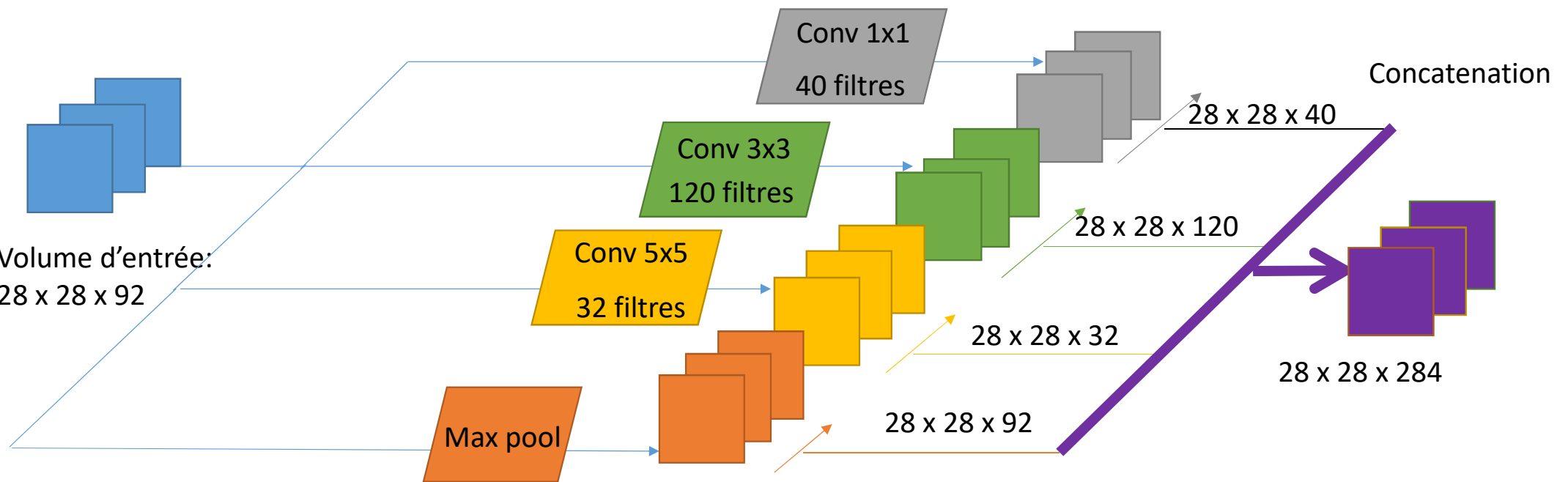
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, **Deep Residual Learning for Image Recognition**, 2015, <https://arxiv.org/abs/1512.03385>

# Deep learning – *GoogLeNet (inception v1)*

**GoogLeNet** /Szegedy & al. 2015/

En général, on ne sait pas trop choisir la taille et le type de couche à utiliser.

L'idée d'un module « inception » est de calculer différentes couches en parallèle et laisser l'algorithme d'apprentissage gérer les couches via le choix des poids

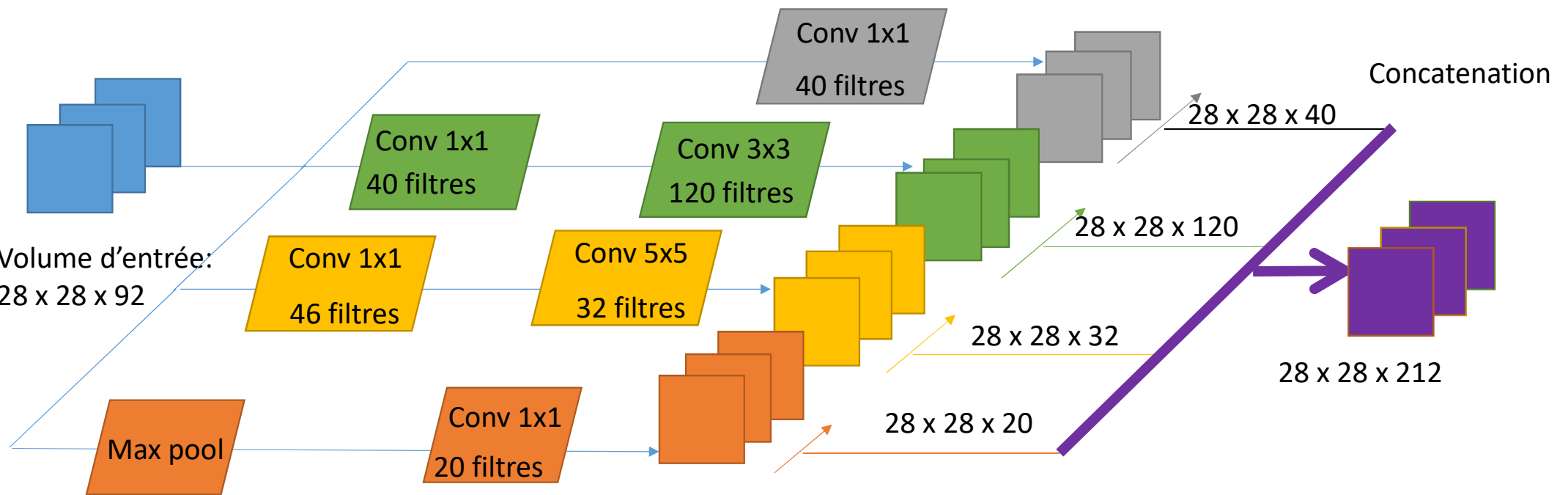


# Deep learning – GoogleNet (*inception v1*)

**GoogLeNet** /Szegedy & al. 2015/

En pratique l'approche précédente requiert énormément d'opérations de calcul.

Une réduction est faite en introduisant des convolution 1x1 afin de réduire le nombre de features



# Deep learning – *GoogLeNet (inception v1)*

**GoogLeNet** /Szegedy & al. 2015/

Le réseau GoogLeNet est essentiellement une mise en série de module inception.  
Le réseau se termine (comme toujours) pour des couches entièrement connectées  
Et une couche soft-max pour la classification

1 module Inception





## Deep learning – *apprentissage des poids & biais*

En général les problèmes traités par les méthode de deep learning nécessite l'apprentissage de millions de paramètres.

Les techniques utilisées reposent toujours sur l'algorithme de rétropropagation du gradient de l'erreur mais sont adaptées.

La plupart des modifications sont pertinentes et efficace en pratique. Cependant les démonstrations théoriques sont peu nombreuses (car on traite un problème d'optimisation non convexe pour lequel il n'existe quasiment aucun résultat générique).

Réseau	Nombre de paramètres
AlexNet	61 millions
VGG16	138 millions
VGG19	
GoogLeNet	7 millions
Inception v3	
ResNet-50	25,5 millions
ResNet-101	